



**Rapport de Soutenance
Genesis
NOVA**

Aurelien FOURÉ
Timoteo CEDOLIN
Lucas BOURGEOIS
Noé BRÉDIF

2025 - 2026

Table des matières

1	Introduction	3
1.1	Présentation du projet	3
1.2	Présentation de l'équipe	3
2	Pour la soutenance 1	5
2.1	La partie graphique	5
2.1.1	L'importance de l'interface dans le projet . . .	5
2.1.2	Conception de l'interface avec Glade	5
2.2	Fonction Solver	8
2.3	Suppression des couleurs	9
2.3.1	Transformation en niveaux de gris	9
2.3.2	Binarisation de l'image : méthode d'Otsu . . .	9
2.4	Prétraitement	10
2.4.1	Morphologie : ouverture et fermeture	10
2.4.2	Enlever le bruit	11
2.5	Détection des composantes connexes	12
2.6	Fonction logique XNOR	13
2.6.1	Fonctionnement réseau neuronal	13
2.6.2	Retour sur notre fonction XNOR	13
2.6.3	L'entraînement du réseau	15
3	Prochaine Soutenance	16

1 Introduction

1.1 Présentation du projet

Le but de ce projet est de concevoir un OCR (*Optical Character Recognition*) capable d'extraire automatiquement les informations contenues dans une image représentant une grille de mots cachés. Le programme doit non seulement identifier la grille et les lettres qu'elle contient, mais aussi détecter la liste des mots à retrouver.

Une fois ces éléments extraits, le système devra être en mesure de rechercher chaque mot dans la grille et d'en déterminer les coordonnées de début et de fin, afin d'indiquer précisément leur position ou, le cas échéant, signaler les mots non trouvés.

L'application disposera d'une interface utilisateur intuitive, permettant à l'utilisateur de sélectionner une image depuis ses dossiers. Après traitement, les résultats s'afficheront directement dans l'interface, présentant les mots détectés, leur statut (trouvé ou non trouvé) ainsi que, pour ceux identifiés, leurs coordonnées exactes dans la grille.

1.2 Présentation de l'équipe

Ce projet est développé par l'entreprise Genesis, qui en assurera la conception et la réalisation. Le produit final portera le nom de NOVA – Neural Optical Vision Analyzer, un acronyme évoquant à la fois la vision artificielle et la puissance d'analyse neuronale mise en œuvre dans le traitement de l'image. Ce nom reflète l'ambition du projet : créer un outil performant, intelligent et autonome capable d'analyser visuellement des grilles de mots cachés avec une grande précision.

Le projet NOVA – Neural Optical Vision Analyzer est développé par une équipe de quatre membres : Aurélien, Timoteo, Noé et Lucas.

Chacun possède un rôle bien défini afin d'assurer une progression efficace et équilibrée du projet.

- Aurélien est en charge de la partie solver, c'est-à-dire de la conception de l'algorithme permettant de rechercher automatiquement les mots dans la grille à partir des données extraites.
- Timoteo s'occupe du développement du réseau neuronal, dont le but est d'améliorer la reconnaissance des lettres et des zones de texte dans l'image.
- Noé travaille sur la détection des éléments visuels : il met en place les méthodes permettant d'identifier la grille, les lettres, ainsi que la zone des mots à trouver.
- Lucas est responsable de la création de l'interface utilisateur, qui permettra de charger une image, d'afficher les résultats et de rendre l'application simple et agréable à utiliser.



Aurélien Fouré



Timoteo Cedolin



Noé Brédif



Lucas Bourgeois

Grâce à cette répartition claire des tâches, chaque membre peut se concentrer sur son domaine de compétence tout en collaborant avec les autres. Cette organisation favorise une synergie d'équipe efficace et permet d'avancer de manière cohérente vers un objectif commun : la réalisation d'un OCR complet et fonctionnel pour l'analyse automatique de grilles de mots cachés.

2 Pour la soutenance 1

2.1 La partie graphique

2.1.1 L'importance de l'interface dans le projet

L'interface graphique joue un rôle central dans ce projet, car elle constitue le principal point de contact entre l'utilisateur et le programme. Sans une interface claire, intuitive et réactive, même l'application la plus performante sur le plan technique risque d'être perçue comme complexe ou inutilisable. Dans le cadre de notre projet, l'interface n'est pas simplement un outil visuel : elle est la passerelle qui permet de rendre accessibles et compréhensibles les différentes fonctionnalités de traitement d'image, comme l'importation, la rotation et la conversion.

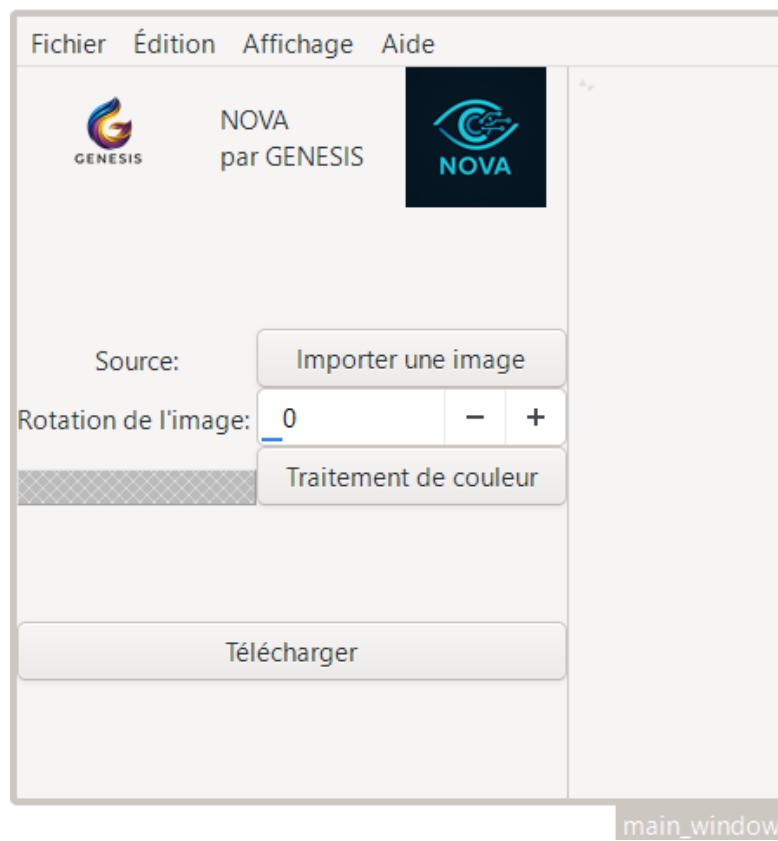
Elle a été pensée pour guider l'utilisateur tout au long du processus, depuis le chargement de l'image jusqu'à la sauvegarde du résultat. L'objectif principal est d'offrir une expérience fluide, où chaque action (cliquer sur un bouton, modifier un angle, lancer une conversion) se traduit immédiatement par un retour visuel clair. En d'autres termes, l'interface transforme des opérations techniques internes (gestion de fichiers, calculs de transformation, rendu graphique) en interactions simples et naturelles.

Grâce à cette approche, le programme devient accessible non seulement à des développeurs ou des étudiants en informatique, mais aussi à des utilisateurs non techniques souhaitant manipuler facilement des images sans connaître les détails de l'implémentation sous-jacente.

2.1.2 Conception de l'interface avec Glade

Pour concevoir l'interface graphique, nous avons utilisé Glade, un éditeur visuel dédié au développement d'interfaces GTK. Cet outil présente un avantage majeur: il permet de séparer clairement la logique du programme (le code en C) de la conception visuelle (l'organisation des éléments à l'écran).

Grâce à Glade, il est possible de créer la fenêtre principale, de positionner les boutons, les zones d'affichage d'image, les champs numériques et tous les autres composants sans écrire une seule ligne de code graphique. Cette séparation entre l'interface et le code permet un développement plus propre et plus modulaire. Le fichier généré par Glade, au format .glade (fichier XML), décrit toute la structure visuelle de l'application : les widgets, leur hiérarchie, leurs propriétés et leurs identifiants.



Dans le code C, nous chargeons ensuite cette interface à l'aide de la fonction `gtk_builder_add_from_file()`, ce qui nous permet de récupérer chaque élément (boutons, images, spinbuttons, etc.) par son identifiant et de lui associer des fonctions spécifiques via des signaux (`g_signal_connect()`).

L'utilisation de Glade apporte également un gain considérable en termes de maintenabilité et d'évolutivité. Si des modifications d'apparence sont nécessaires par exemple changer la disposition des boutons ou ajouter un nouveau champ, elles peuvent être réalisées directement dans Glade, sans modifier le code source. Cela rend la mise à jour de l'interface beaucoup plus rapide et réduit les risques d'erreurs logicielles. Enfin, cette approche facilite la collaboration entre les membres du projet : un développeur peut se concentrer sur la logique du programme tandis qu'un autre travaille sur l'apparence et l'ergonomie de l'application, tout en conservant une cohérence globale.

2.2 Fonction Solver

Pour ma part, j'ai commencé par développer la fonction solver, dont le rôle est de lire une grille contenue dans un fichier texte ainsi que le mot à rechercher, puis de retourner les coordonnées de départ et de fin du mot dans la grille.

Cette première étape s'est révélée relativement simple, car elle repose sur un algorithme de recherche séquentielle dans un tableau généré à partir du fichier texte. Il s'agit essentiellement de parcourir la grille dans toutes les directions possibles (horizontale, verticale et diagonale) jusqu'à repérer la séquence de lettres correspondant au mot recherché. En quelques jours, cette partie du projet a pu être finalisée sans réelle difficulté technique.

```
int Check(int x,int y, int dx, int dy, char* word,int rows , int cols, char**grid)
{
    int x1 = x;
    int y1 = y;
    int i = 1;
    while(word[i] != '\0')
    {
        x += dx;
        y += dy;
        if(x < 0 || y < 0 || y >= rows || x >= cols)
            return 1;
        if(grid[y][x] != word[i])
            return 1;
        i++;
    }
    printf("(%d,%d)(%d,%d)\n",x1,y1,x,y);
    return 0;
}

aurel@pcsirow:/mnt/c/Users/aurel/Documents/Epita/OCR/Projet_Ocr/Solver$ cat grid
HORIZONTAL
DXRAHCLBGA
DIKCILEOKC
IGAJHYLYHI
HGFGODTIOT
GDLROWKBFR
PLNRDNERGE
JHAIDUAJGV
UKGFFOLLEH
aurel@pcsirow:/mnt/c/Users/aurel/Documents/Epita/OCR/Projet_Ocr/Solver$ ./solver grid horizontal
(0,0)(9,0)
aurel@pcsirow:/mnt/c/Users/aurel/Documents/Epita/OCR/Projet_Ocr/Solver$ ./solver grid hello
(9,8)(5,8)
aurel@pcsirow:/mnt/c/Users/aurel/Documents/Epita/OCR/Projet_Ocr/Solver$ ./solver grid rien
Not found
aurel@pcsirow:/mnt/c/Users/aurel/Documents/Epita/OCR/Projet_Ocr/Solver$ |
```

2.3 Suppression des couleurs

2.3.1 Transformation en niveaux de gris

Pour transformer l'image en niveaux de gris, il faut parcourir tous les pixels.

Un pixel a plusieurs composantes dont le rouge, le vert et le bleu.

Il peut y en avoir d'autre, mais elle ne sont pas utilisée pour le projet.

Chaque pixels doit calculer une moyenne pondérée des trois premières composantes.

Puis on affecte le gris obtenu aux composantes de l'image.

2.3.2 Binarisation de l'image : méthode d'Otsu

La binarisation utilise un seuil calculé automatiquement selon la méthode d'Otsu.

Le but est transformer l'image avec les couleurs extremums :

Si le pixel est plus clair que le seuil alors il devient blanc, sinon il devient noir.

Les poids ω_i représentent la probabilité d'être dans la i -ème classe, chacune étant séparée par un seuil t . Finalement, les σ_i^2 sont les variances de ces classes.

$$\begin{aligned}\sigma_w^2(t) &= \omega_1(t)\sigma_1^2(t) + \omega_2(t)\sigma_2^2(t) \\ \sigma_b^2(t) &= \sigma^2 - \sigma_w^2(t) = \omega_1(t)\omega_2(t) [\mu_1(t) - \mu_2(t)]^2\end{aligned}$$

tOtsu montre que minimiser la variance intra-classe revient à maximiser la variance inter-classe. Elle est exprimée en termes des probabilités de classe ω_i et des moyennes de classes μ_i qui à leur tour peuvent être mises à jour itérativement.

On calcule l'histogramme et les probabilités de chaque niveau d'intensité et on défini les $\omega_i(0)$ et $\mu_i(0)$ initiaux.

Pour chaque seuils possibles $t \in [[0; 256[$, on mette à jour ω_i et μ_i et on calcule $\sigma_b^2(t)$.

Le seuil désiré correspond au maximum des $\sigma_b^2(t)$.

2.4 Prétraitement

2.4.1 Morphologie : ouverture et fermeture

En Théorie, une figure est un amas de même valeur non nulle. En pratique, les écritures sont en noir sur fond blanc, un pixel sans couleur (en noir) a une valeur nulle, alors que un pixel coloré (plus claire) a une valeur non nulle.

Pour les opérations morphologiques suivantes, on garde la définition mathématique.

Donc on considérera que les figures sont les zones claires. (le fond, les compteurs (trous des lettres), dessins clairs sur un objet foncé, mais pas les lettres !)

$X \subset [[0; w[[\times [[0; h[= \{(0, 0), (0, 1), \dots (w, h)\}$ l'ensemble des coordonnées d'une figure.

$$B = \{-1, 0, 1\}^2 = \{(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 0), (0, 1), (1, -1), (1, 0), (1, 1)\}$$

Pour un pixel donné, B représentante les 4 cotés, les 4 diagonales et lui même.

La zone autour d'un pixel est donc au maximum de $|B| = 3 \times 3 = 4 + 4 + 1 = 9$ pixels.

Le translaté de B positionné sur un pixel p est $B_p = \{p + b \mid b \in B\}$.

Les exemples considère une main comme figure.

- Dilatation : élargir la figure.

Pour chaque pixel p , On prend la plus grande valeur (maximum) autour du pixel p et on l'affecte à p . (exemple : moufle)

$$\delta_B(X) = X \oplus B = \{p + b \mid b \in B, p \in X\} = \bigcup_{p \in X} B_p$$

- Érosion : rétrécir la figure.

Pour chaque pixel p , On prend la plus petite valeur (minimum) autour du pixel p et on l'affecte à p . (exemple : paume d'une petite main)

$$\epsilon_B(X) = X \ominus B = \{p \mid B_p \subset X\} = \bigcap_{b \in B} X_{-b}$$

- Ouverture : enlever les débordements.

La figure rétrécit puis s'agrandit à nouveau. On perd les pixels qui n'ont pas touché un pixel qui touche des 8 côtés la figure. (exemple : main sans doigts)

$$\gamma_B(X) = X \circ B = (X \ominus B) \oplus B = \delta_B(\epsilon_B(X))$$

- Fermeture : ajouter les encerclément.

La figure s'agrandit puis rétrécit à nouveau. On gagne les pixels qui touche des 8 côtés un pixels qui touche un pixel de la figure. (exemple : main palmée)

$$\phi_B(X) = X \bullet B = (X \oplus B) \ominus B = \epsilon_B(\delta_B(X))$$

- Médiane :

Pour chaque pixel p de l'image, On prend et trie les valeurs des pixels autour de p puis on récupère celui du milieu et on l'affecte à p .

(exemple : une main palmée avec des doigt très court)

- Tris puis choix :

On peut la considérer un paramètre a pour connecter les 5 algorithmes précédents.

Pour l'érosion $a = 0$, pour la dilatation $a = 1$ et pour la médiane $a = 0.5$.

Il est alors possible de mettre ce paramètre à n'importe quel valeur dans $[0; 1]$.

On affecte à chaque pixel p la $\lfloor (nombre_de_pixels_autour_de_p-1) \times a \rfloor$ -ième valeur de la liste triée des pixels autour de p (indexation basée sur 0).

On a recréé alors l'ouverture et la fermeture morphologique avec ce nouveau paramètre pour affiner le traitement.

2.4.2 Enlever le bruit

- Valeur moyenne :

Pour chaque pixel p de l'image, On somme les valeurs des pixels à une certaine distance de p , puis on divise par le nombre de pixels rencontrés et on affecte le résultat à p .

- Suppression brute des petites tache de pixels :

On supprime les tache de pixels de couleurs foncée lorsque le nombre de pixels de la tache est sous un certain seuil. On utilise un parcourt en largeur avec une 8-connectivité pour que les diagonales soit comptée comme dans une même tache. Si la taille de la tache est au dessous du seuil, alors on dépile tous les pixels de cette tache et on les met en blanc.

2.5 Détection des composantes connexes

La détection automatique des éléments présents dans l'image : L'objectif était d'identifier et d'extraire différentes zones : la grille elle-même, les lettres qui la composent, la zone listant les mots à trouver, ainsi que les mots et leurs lettres respectives.

Cette partie s'est avérée bien plus délicate, car il existe de nombreuses approches possibles, traitement d'image, reconnaissance de formes, filtrage, etc. mais notre contrainte principale était de n'utiliser que la librairie standard, ce qui a considérablement restreint les options disponibles. Sans bibliothèques spécialisées comme OpenCV ou PIL, certaines méthodes classiques de traitement d'image devenaient beaucoup plus ardues à mettre en œuvre.

Avec Noé, nous avons donc pris le temps de réfléchir à la stratégie la plus adaptée dans ce contexte limité. Après plusieurs essais et discussions, nous avons opté pour une méthode basée sur la détection de densité pour distinguer les zones pertinentes de l'image.

L'idée était ensuite de repérer la plus grande composante connexe (avec une 8-connectivité), correspondant à la grille principale, afin de pouvoir isoler et traiter les autres éléments plus facilement. Cette approche, bien qu'un peu rudimentaire, nous a permis d'obtenir des résultats satisfaisants sans avoir recours à des outils externes.

2.6 Fonction logique XNOR

2.6.1 Fonctionnement réseau neuronal

Un réseau neuronal fonctionne de la même manière qu'un vrai cerveau, c'est à dire qu'il a une couche d'entrée, une couche cachée et une couche de sortie.

La couche d'entrée est notre point de départ dans le système, dans notre exemple de la fonction XNOR, cela représente les entrées A et B ainsi que les valeurs qu'elles ont.

La couche cachée, quant à elle, permet de faire les calculs du code, c'est la partie où le neurone "réfléchit" ce qui est demandé pour ensuite transmettre les données calculées à la couche de sortie.

Enfin, la couche de sortie donne la décision finale donc dans notre exemple elle permet de dire si l'opération logique donne le résultat binaire 0 ou 1.

Aussi, le réseau neuronal comporte des biais et des poids qui ont chacun leur importance. Les poids servent pour déterminer l'importance de chaque connection entre les neurones. Les biais permettent d'améliorer l'ajustement du modèle. On peut faire une analogie avec l'étude d'un produit sur le marché avec comme critère les poids et comme condition minimal le biais. On peut faire un exemple avec l'achat d'une voiture où chaque poids n'est pas forcément le même (prix, couleur, marque) et le biais (condition minimal à respecter) pour la voiture en question.

2.6.2 Retour sur notre fonction XNOR

Dans notre fichier, on a donc plusieurs fonctions qui font chacun différentes choses. Pour commencer, on définit un struct avec ce dont on a besoin, une fonction qui initialise les valeurs de ce qu'on a dans le struct relié à un pointeur. Ensuite, une fonction qui fait les calculs de la fonction logique XNOR dans laquelle la couche cachée et la couche de sortie sont apparentes.

On a aussi une fonction qui est le point le plus important du réseau, autrement dit la fonction d'activation. C'est celle qui permet de faire les calculs pour avoir des valeurs aussi proche de 0 ou de 1 possible pour avoir une bonne représentation binaire. Cette fonction est donc appelée plusieurs fois au moment de faire les calculs dans la couche cachée et la couche de sortie. C'est la fonction Sigmoid qui est représenté par $f(x) = \frac{1}{1+e^{-x}}$

```
Nombre de simulation d'entraînement (entre 1 et 100 000): 5000
```

```
== Début de l'entraînement ==
simulation 0: Loss = 0.240518
simulation 1000: Loss = 0.179243
simulation 2000: Loss = 0.004658
simulation 3000: Loss = 0.001757
simulation 4000: Loss = 0.001062
simulation 4999: Loss = 0.000755
== Entrainement terminé ==
```

```
Poids APRES l'entraînement:
```

```
== Architecture du Réseau ==
```

```
Couche d'entrée: 2 neurones (A, B)
```

```
Couche cachée: 2 neurones
```

```
    Neurone 1:
        w1=-6.1988, w2=6.2075, bias=-3.4338
    Neurone 2:
        w1=-5.2708, w2=5.5258, bias=2.6114
```

```
Couche de sortie: 1 neurone
```

```
        w1=-8.5530, w2=8.3600, bias=-3.9579
```

```
Résultats APRES l'entraînement:
```

```
== Table de Vérité !A.!B + A.B ==
A | B | !A.!B | A.B | Attendu | Résultat | Correct
---|---|-----|-----|-----|-----|-----
0 | 0 |     1 |     0 |     1 |     1 |     ✓
0 | 1 |     0 |     0 |     0 |     0 |     ✓
1 | 0 |     0 |     0 |     0 |     0 |     ✓
1 | 1 |     0 |     1 |     1 |     1 |     ✓
```

```
Précision: 4/4 (100%)
```

2.6.3 L'entraînement du réseau

Afin d'entraîner le modèle à calculer la fonction XNOR, on dispose de trois éléments importants qui sont la fonction d'entraînement, la fonction qui permet l'apprentissage et la fonction qui calcule les erreurs et retourne un résultat pénalisant de sorte à faire moins d'erreur.

La fonction du calcul d'erreur est faite de sorte à ce que les erreurs aient de hautes valeurs pour que les grosses erreurs soient très pénalisées (erreur quadratique). Notre fonction d'entraînement est simplement une boucle for dans une autre où l'on applique la rétropropagation, calcule les pertes totales et enfin afficher ces pertes qui convergent vers 0.

Enfin, la rétropropagation qui est l'algorithme d'apprentissage se composent de six étapes. Premièrement, on calcule toutes les valeurs de sortie donc les différents tests de la fonction à faire. Ensuite, il calcule l'erreur de sortie pour savoir si le modèle a ou non fait des erreurs. Il ajuste ensuite le signal d'erreur pour savoir vers quel signe ajuster les poids. Il faut ensuite propager l'erreur aux neurones cachés. Enfin, il met à jour les poids en faisant le produit de l'ancien poids, du signal d'erreur et de la valeur prise par A et B dans notre fonction XNOR.

Ces éléments sont donc les plus importants pour entraîner notre modèle.

3 Prochaine Soutenance

Pour la prochaine soutenance, plusieurs objectifs majeurs ont été fixés afin d'aboutir à une version complète et pleinement fonctionnelle du projet NOVA – Neural Optical Vision Analyzer. Ces étapes représentent la finalisation de l'ensemble des modules principaux du système, depuis le traitement de l'image jusqu'à l'affichage des résultats dans l'interface.

Tout d'abord, nous devons mettre en place le prétraitement complet des images, comprenant notamment la rotation automatique. Cette étape est essentielle pour corriger les éventuelles inclinaisons de la grille dans l'image source et garantir une lecture optimale par le réseau neuronal. Ce module assurera ainsi une standardisation des images avant leur analyse.

Ensuite, le réseau de neurones devra être finalisé et rendu totalement fonctionnel. Il s'agira d'achever la phase d'apprentissage du modèle afin qu'il puisse reconnaître avec précision les lettres de la grille ainsi que celles de la liste des mots à trouver. Cette étape, véritable cœur intelligent du projet, permettra à l'application d'interpréter automatiquement le contenu visuel sans intervention humaine.

Une fois la reconnaissance effectuée, nous devrons passer à la reconstruction complète de la grille et de la liste de mots à partir des résultats du réseau neuronal. L'objectif est de reconstituer fidèlement les informations extraites afin de pouvoir les exploiter dans la phase suivante.

Vient ensuite la résolution de la grille, qui consiste à appliquer le module “solver” pour rechercher la position de chaque mot dans la grille reconstruite. Cette étape combinera les résultats du réseau neuronal et les algorithmes de recherche afin d'identifier les mots trouvés et leurs coordonnées exactes.

Une fois la résolution terminée, il sera nécessaire de développer l'affichage de la grille dans l'application, permettant à l'utilisateur de visualiser clairement les mots identifiés, leurs positions, et le résultat global de l'analyse.