PROJET OCR CLAPS

Rapport de projet

Coureau Adrien
Del-Pozo Corentin
Fournier Pierre-Antoine
Lou Sophie

Table des matières

1 Introduction

Tout au long du projet CLAPS Word Search OCR, la présence d'un rapport de projet est primordiale. Il nous permettra de nous concentrer sur le bilan de ce qui a été fait et la manière dont nous allons traiter le projet sur le temps imparti (du mois de septembre au mois de décembre). Nous présenterons ainsi l'état d'avancement de notre projet, ainsi qu'un compte rendu qui détaillera les tâches prévues dans le cahier des charges.

L'objectif de ce projet est de nous offrir la possibilité de mettre en oeuvre de manière concrète, les connaissances acquises en cours. Cela nous donnera ainsi l'occasion d'augmenter nos acquis personnels nécessaires au projet.

Le but final de ce projet est de réaliser un logiciel de type OCR qui résout une grille de mots cachés en 3 mois. Notre application prendra donc en entrée une image représentant une grille de mots cachés et affichera en sortie la grille résolue.

Ainsi, nous avons découpé ce projet en plusieurs parties :

— Prétraitement de l'image

— Reconnaissance des caractères (Réseau de neurones)

— Résolution d'une grille de mots cachés (Solver)

— Interface graphique

2 Répartition des Tâches

Tâches	Pierre-Antoine	Corentin	Adrien	Sophie
Interface Utilisateur			✓	
Niveau de gris	✓			
Réduction de bruits	✓			
Rotation				✓
Binarisation			✓	
Détection de Grille		✓		
Solver				✓
Réseaux de Neurones		✓		

3 Avancement

Tâches	09/10/2024	09/12/2024
Interface Utilisateur	60%	100%
Niveau de gris	100%	100%
Réduction de bruits	50%	100%
Rotation	50%	100%
Binarisation	100%	100%
Détection de Grille	70%	100%
Solver	100%	100%
Réseaux de Neurones	50%	100%

Toutes les tâches ont été réalisées dans le temps imparti. Le projet est fonctionnel.

4 Notre Equipe

4.1 Lou Sophie

Animée par mon attirance pour les mathématiques, la réalisation de la rotation manuelle d'image, m'a permis de nourrir cette curiosité, par l'application à des choses concrètes et visuelles (ici, des grilles de mots mêlées). Le fait d'avoir l'occasion de mettre en pratique des connaissances mathématiques, pouvant paraître abstraites, en l'associant à de l'informatique, m'a rappelé l'importance de ces deux domaines.

De plus, en mettant en place un algorithme de Solver, cela m'a permis de comprendre l'implémentation d'un algorithme de résolution de mots cachés. Ensemble, nous avons pu déterminer quel était le plus efficace à implémenter. Toujours en quête d'application concrète, le fait de savoir de manière immédiate si un mot a été trouvé ou non, était très encourageant.

4.2 Fournier Pierre-Antoine

Pour ce projet, j'ai choisi de faire les niveaux de gris et la réduction de bruit car le traitement d'image m'intéresse. De plus, j'envisage de faire la majeur des images à EPITA. Ce projet est donc un bon moyen de découvrir si le traitement d'images me plaît.

4.3 Del-Pozo Corentin

Pour ce projet, j'ai choisi de m'occuper du réseau de neurones, car c'est un sujet en informatique qui me passionne beaucoup. Avant de commencer le projet, je m'étais déjà renseigné sur son fonctionnement et sur toutes les possibilités qu'il offre. Le développement du réseau de neurones m'a permis d'approfondir mes connaissances en apprentissage supervisé et d'appliquer des concepts théoriques à des problèmes pratiques. Cela m'a également permis de mieux comprendre comment les algorithmes peuvent apprendre et s'adapter à des données variées, ce qui est essentiel dans le contexte de l'intelligence artificielle.

En ce qui concerne la détection de la grille, j'ai choisi de m'occuper de ce problème parce qu'au début, notre groupe n'avait aucune idée de quelle direction prendre. J'apprécie particulièrement les défis qui consistent à chercher des solutions dans un environnement où nous disposons de peu d'indices et de ressources. Cette contrainte nous pousse à développer nos propres solutions et à faire preuve de créativité, ce qui rend le processus d'apprentissage d'autant plus stimulant.

4.4 Coureau Adrien

Passionné d'informatique depuis petit, l'univers de la création m'attire beaucoup (dessin, dessin digital, production de musique). J'ai choisi de développer l'interface du projet en gtk3 avec l'aide de Glade parce que j'aime créer un produit fini et cohérent à partir de zéro.

La conception d'une interface facile à prendre en main est cruciale pour fluidifier l'expérience de l'utilisateur ainsi que renforcer la rapidité des tâches et des tests pour les développeurs. Elle est la liaison entre ce que l'on voit, les boutons, et toutes les actions que peut effectuer notre application d'OCR.

5 Pourquoi le format BMP

Nous avons choisi ce format car il rend plus simple le parcours d'une image, tout en conservant une bonne qualité avec ce format. Cependant, il est plus volumineux que le format `PNG`, il est donc important de supprimer toutes les images en `BMP` que nous avons créées avant un éventuel crash. Pour cela, un script suffit. Nous devons aussi convertir l'image fournie par l'utilisateur avant d'effectuer toute opération.

5.1 Parcourir une image BMP

Pour faire un parcours d'une image, il nous suffit d'appeler la fonction

```
1    SDL_LoadBMP("img.bmp")
```

qui renvoie une `surface`. La `surface` est une structure de la librairie `SDL` qui contient la hauteur, la largeur et le pointeur vers le tout premier pixel de l'image.

L'image est contenue dans un seul tableau : les 3 premières valeurs sont les pixels rouge, vert, et bleu du premier pixel, puis les 3 suivantes ceux du second et ainsi de suite.
Pour parcourir une image nous utilisons le code suivant :

```
1  for (int i = 0; i < height; i++)
2  {
3      for (int j = 0; j < width; j++)
4      {
5          Uint8* pixel = pix + j * pitch + i * bpp;
6          // do stuff on the image
7      }
8  }
```

Nous avons une double boucle plutôt classique. Ensuite, pour récupérer un pointeur vers la couleur rouge du pixel actuel, il nous suffit de faire le calcul suivant :
`pix` est le pointeur du tout premier pixel de l'image, accessible avec `surface→pixels`. On lui ajoute `i` qui correspond à la rangée horizontale actuelle multipliée par le `pitch` (accessible via `surface→pitch`). Le `pitch` est le nombre d'octets entre les lignes. Il n'est pas toujours égal au nombre de pixels multiplié par le nombre d'octets par pixel (`bpp`), car des octets peuvent être ajoutés en bout de ligne pour des raisons de performance. Ainsi, `pix + i * pitch` nous permet de récupérer la bonne ligne de pixels de l'image. Nous ajoutons `j`, correspondant au `j`-ème pixel de la `i`-ème rangée, et nous multiplions `j` par le nombre d'octets par pixel (`surface→format→BytesPerPixel`). Dans notre cas, `BytesPerPixel` vaut 3.

6 Réduction de Bruit

Dans ces images données dans le cahier des charges, le niveau 2 contient des images avec du bruit. Nous avons implémenté 2 algorithmes pour réduire le bruit : le Flou Médian et le Flou Gaussien. Cependant nous n'avons pas obtenu de résultat suffisant avec 2 algorithmes. Nous avons donc arrêté d'utiliser le flou gaussien, améliorant le flou médian. Le plus gros changement est cependant les algorithmes de contraste d'image.

6.1 Le Flou Gaussien

Le flou Gaussien utilise également un noyau, mais il est prédéfini. On peut choisir sa taille (3x3, 5x5, . . .), et dans notre cas nous utilisons un noyau de taille 3x3. Pour calculer les valeurs du noyau, nous appliquons la fonction Gaussienne, qui produit une courbe similaire à l'image fournie (elle peut être plus pentue ou plus allongée).

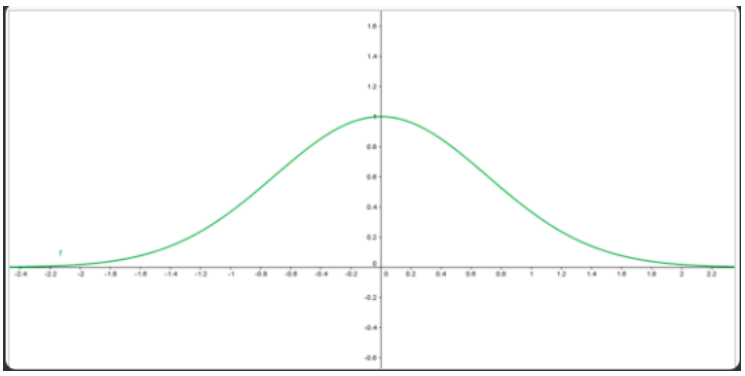


FIGURE 1 – Courbe de Gauss

Les valeurs fournies à la fonction Gaussienne, qui prend trois arguments, sont X et Y, représentant respectivement les coordonnées (x, y) des pixels autour du milieu du noyau de coordonnées (0,0). Le troisième argument est sigma, qui permet d'ajuster la luminosité de l'image. Pour déterminer le sigma optimal, nous avons testé plusieurs valeurs croissantes pour trouver la meilleure. La fonction est la suivante :

$$\mathcal{G}(x,y) = \frac{1}{2\pi\sigma^2} \exp^{-\frac{x^2+y^2}{2\sigma^2}} \text{ et } \sigma \in \mathbb{R}$$

```
#define M_PI        3.14159265358979323846

double gFunc(int x, int y)
{
    const double sig = 0.58;
    double part1 = 1 / (2 * M_PI * sig * sig);
    double power = -(x * x + y * y) / (2 * sig * sig
       );
    double part2 = exp(power);
    return part1 * part2;
}
```

6.2 Le Flou Médian

Comme pour le flou gaussien, nous avons besoin d'un noyau. Cependant le noyau n'est pas constant. Dans notre projet, pour avoir de bon résultat, le noyau est de taille 3x3. Nous mettons dans le noyau les 9 valeurs autour du pixel actuel. Nous trions ensuite cette liste et donnons la valeur du milieux du tableau (la 4ème). Cet algorithme permet de faire disparaître entièrement les images contenant un bruit assez léger.

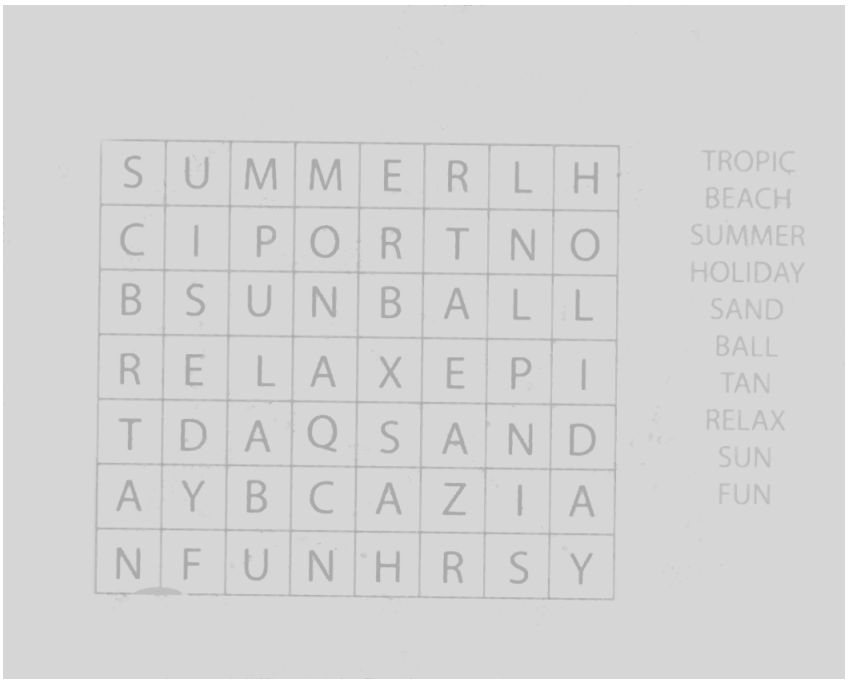


FIGURE 2 – Flou Médian

6.3 Contraste

Avant nous diminuons juste la couleur du pixel si il respecte cette condition.

```
1 // side_pixel is an array containing the 8 pixels
    around a pixel
2 if (side_pixel[k] < pixel[0] && pixel[0] < 200) { /*
    ...*/ }
```

* Nous avons maintenant 2 algorithmes de contraste.

6.3.1 Contraste par étirement

Ce contraste est très simple, nous cherchons la valeur minimum dans l'image, c'est-à-dire le pixel qui a la plus petite valeur (le plus foncé possible). Nous faisons de même pour la valeur la plus élevée (la plus claire). Puis nous parcourons tous les pixels de l'image et leur appliquons la formule suivante :

$$I_{new} = \frac{I - I_{min}}{I_{max} - I_{min}} * 255$$

Ce qui nous donne l'image suivante :

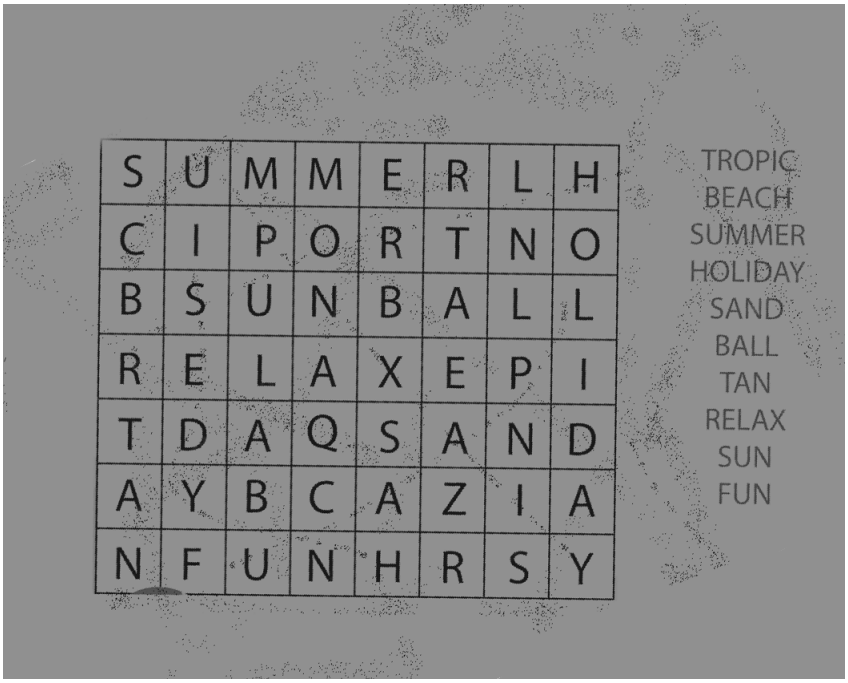


FIGURE 3 – Contraste par étirement

6.3.2 Contraste "dur"

Pour cet algorithme l'objectif est d'accentuer fortement les pixels clairs en les rapprochant plus du blanc, on fait de même pour les pixels noirs. Nous prenons le point médian 128 comme nos pixels sont des tableaux de 3 Uint8. Ensuit pour chaque pixel :

— On lui soustrait le point médian
— On multiplie le résultat par un facteur de contraste. Plus le facteur est grand plus les différence de couleur seront accentuer. Dans notre cas, nous avons pris 2 comme facteur.
— Puis nous ajoutons le point médian au résultat de l'étape précédente ce qui nous donne la nouvelle valeur du pixel.

Ce procédé nous donne des images comme celle-ci :

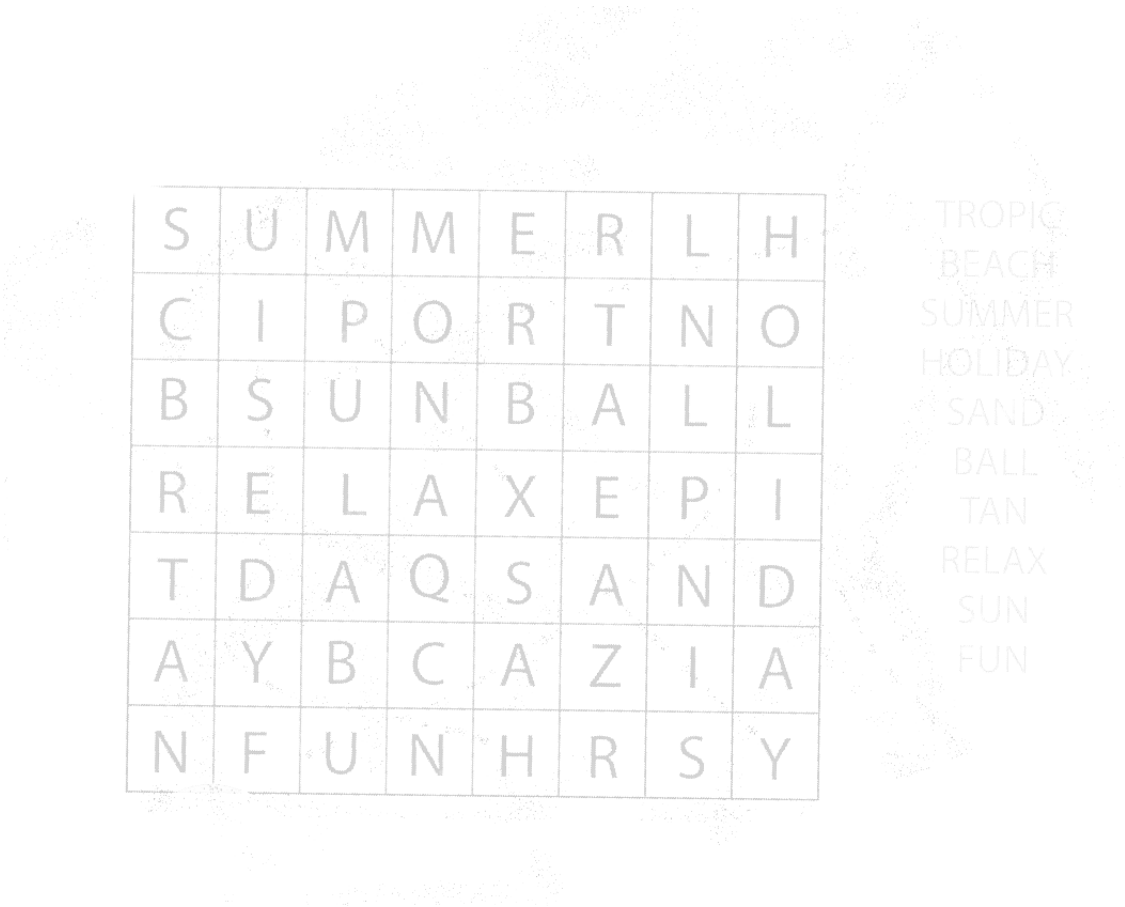


FIGURE 4 – Contraste "dur"

6.4 Résultat

Une fois tous les algorithmes précédents appliqués, nous obtenons une image de cette qualité.

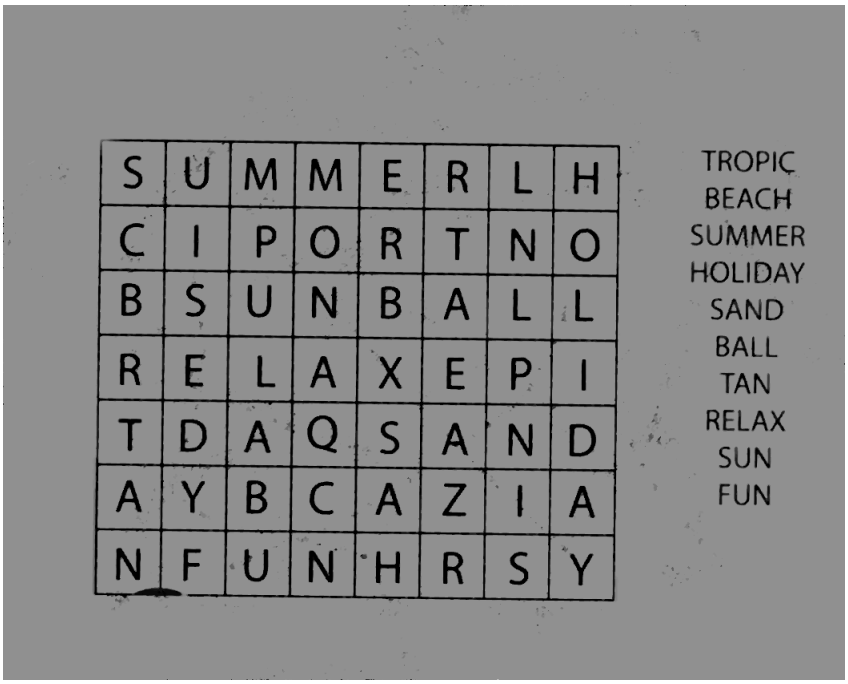


FIGURE 5 – Résultat Final

Comme nous pouvons le voir, le bruit a quasiment disparu de l'image. Cependant cette image n'est pas utilisable car lorsque nous appliquons la binarisation, certaines parties des lettres sont trop claires et les groupes de formes ne sont donc pas bien détectés. Par manque de temps, nous n'avons pas modifié la binarisation, ainsi, les fonctions de réduction de bruit ne sont pas dans le code principal.

7 Rotation de l'image

Plusieurs méthodes peuvent être appliquées afin d'effectuer une rotation d'image. Voici 2 différentes techniques, aboutissant à celle conservée, étant la plus performante.

7.1 Rotation avec une seul matrice

Cette méthode de rotation consiste à utiliser cette formule :

$$x' = \cos(\theta) * x - \sin(\theta) * y$$
$$y' = \sin(\theta) * x + \cos(\theta) * y$$

Provenant de la représentation matricielle suivante :

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & \cos(\theta) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

(avec θ l'angle en degré)

7.1.1 Problème rencontré

Malgré la simplicité de cette formule, l'application de celle-ci provoque de manière persistante des problèmes d'aliasing, dû à l'apparition non voulue de points noirs, pouvant nuire à la détection des lettres. La rotation avec une seule matrice affecte donc la qualité de l'image, dû à une mauvaise utilisation. En effet, nous sommes partis des coordonnées des pixels de l'image d'origine et nous avons calculé leur position après rotation dans la nouvelle image. Cette technique s'appelle l'échantillonnage en avant, ayant pour effet de laisser des trous dans l'image.

FIGURE 6 – Problème 1

7.2 Rotation par cisaillement

Ne sachant pas que le problème rencontré était dû à une mauvaise utilisation, nous avons ainsi voulu appliquer le principe de rotation par cisaillement (aussi appelé rotation par Shearing), qui utilise l'algorithme de Shearing, prenant chaque pixel de l'image, puis applique 3 multiplications successives, par des matrices avec θ notre angle en radian.
Elle utilise donc cette représentation matricielle :

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} 1 & -\tan(\frac{\theta}{2}) \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ \sin(\theta) & 1 \end{pmatrix} \begin{pmatrix} 1 & -\tan(\frac{\theta}{2}) \\ 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

(avec θ l'angle en degré)

Afin que la rotation puisse s'effectuer dans le sens horaire et non anti-horaire, nous avons ainsi inverser les signes obtenant les formules suivantes :

— 1^{er} cisaillement :

$$x' = x + \tan(\theta) * yy' = y$$

— 2^{eme} cisaillement :

$$x'' = x'y'' = x'' + \sin(\theta) * (-1) * x' * y'$$

— 3^{eme} cisaillement :

$$x''' = x'' + \tan(\theta) * yy''' = y''$$

(avec θ l'angle en degré)

Voici un exemple de représentation avec un angle de 20° :

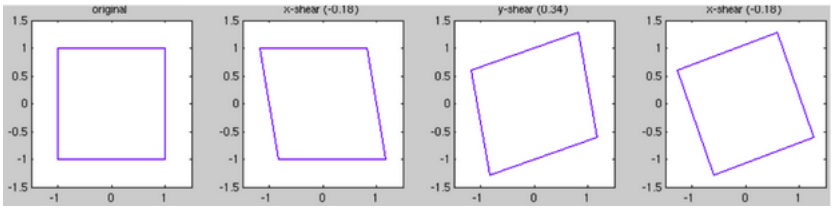


FIGURE 7 – Rotation de 20°

7.2.1 Problème rencontré

Malgré la netteté de l'image, celle-ci affichait des failles, par une perte d'une partie de l'image lors d'angles, comme celui de 35°. En effet, il était difficile de la centrer correctement.

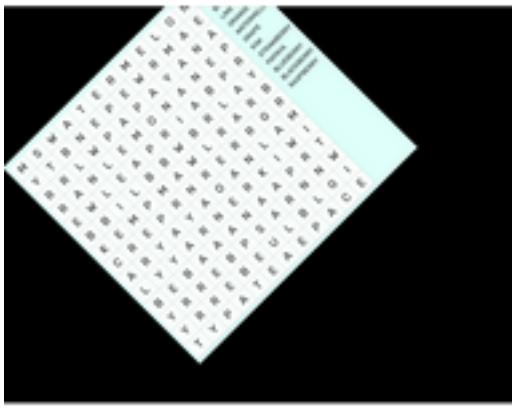


FIGURE 8 – Problème 2

7.3 Rotation avec matrice de rotation

Enfin, cette méthode a été privilégiée par son efficacité, utilisant la formule de “matrice de rotation” vu en 3.1 :

$$x' = \cos(\theta) * x - \sin(\theta) * y$$

$$y' = \sin(\theta) * x + \cos(\theta) * y$$

(avec θ l'angle en degré)

Cependant, au lieu de partir des coordonnées des pixels de l'image d'origine et nous avons calculé leur position après rotation dans la nouvelle image. Nous sommes partis des coordonnées de pixels dans la nouvelle image puis nous avons recherché les pixels correspondants dans l'image d'origine (c'est un ”échantillonnage inverse”), ce qui garantit que chaque pixel de la nouvelle image est couvert et minimise les pertes de pixels.

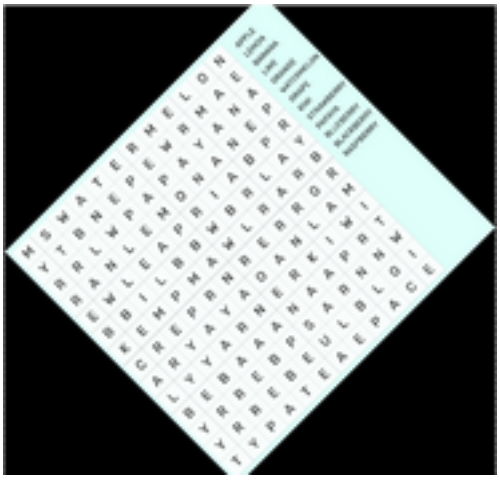


FIGURE 9 – Résultat final

7.4 Optimisation de l'allocation de mémoire pour rotation

L'optimisation de l'allocation de mémoire consiste à calculer précisément la taille minimale nécessaire pour contenir l'image après rotation. Cela implique de déterminer les dimensions du rectangle englobant en utilisant des principes trigonométriques.

7.4.1 Calcul des nouvelles dimensions

Lorsqu'une image rectangulaire est tournée, les nouvelles dimensions du rectangle englobant dépendent de l'angle de rotation. Ces dimensions sont obtenues à partir des projections des côtés de l'image d'origine sur les nouveaux axes après rotation.

7.4.2 Formules utilisées

Les nouvelles dimensions du rectangle sont données par les formules suivantes :

— largeur de la nouvelle dimension :

$$\text{new\_width} = |\text{width} \cdot \cos(\theta)| + |\text{height} \cdot \sin(\theta)|$$

— hauteur de la nouvelle dimension :

$$\text{new\_height} = |\text{width} \cdot \sin(\theta)| + |\text{height} \cdot \cos(\theta)|$$

Ici, width et height désignent respectivement la largeur et la hauteur de l'image originale, tandis que θ représente l'angle de rotation.

7.4.3 Gestion des projections négatives

Comme $\cos(\theta)$ et $\sin(\theta)$ peuvent être négatifs selon le quadrant de l'angle, et que les dimensions physiques d'une image (largeur et hauteur) ne peuvent jamais être négatives, nous utilisons la valeur absolue pour résoudre ce problème. Ainsi, les projections négatives sont corrigées avec la valeur absolue :

$$|\cos(\theta)| \; et \; |\sin(\theta)|$$

7.4.4 Origine mathématique

Les formules ci-dessus découlent directement de la transformation de rotation dans le plan cartésien. Lorsque l'on fait tourner un point ou une figure dans un plan, les coordonnées de chaque point sont transformées de la manière suivante :

$$x' = x \cdot \cos(\theta) - y \cdot \sin(\theta)$$

$$y' = x \cdot \sin(\theta) + y \cdot \cos(\theta)$$

Ces transformations appliquées aux coins du rectangle permettent de déterminer l'espace total requis après rotation.

7.4.5 Avantages de l'optimisation

En optimisant ainsi l'allocation de mémoire, nous avons réduit la consommation de ressources, tout en s'assurant que l'image entière après rotation, est bien incluse dans la nouvelle surface. La différence visuelle, pouvant être minime, elle reste tout de même non négligeable en terme d'allocation de mémoire. En voici un exemple, où nous pouvons observer que la figure 10 non optimisée, prend, après la rotation un espace total supérieur à celui de la figure 11.

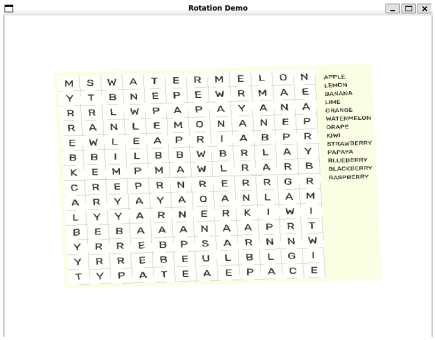


FIGURE 10 – Image après rotation non optimisée

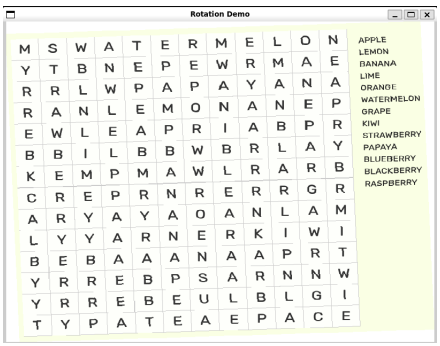


FIGURE 11 – Image après rotation optimisée

8 Rotation automatique

Afin d'effectuer le redressement automatique de l'image, nous avons découpé ce processus, en deux tâches distinctes. Tout d'abord la détection d'angle de rotation a été un premier défi, puis rotation de l'image nous a permis d'appliquer la rotation avec matrice de rotation précédemment écrite.

8.1 Détection de l'angle

La structure Shape a été utilisée pour représenter les caractéristiques géométriques d'un objet (ou d'une image) à manipuler. Nous verrons en profondeur les composantes exactes de la structure Shape dans la détection de grille. Cependant, dans notre cas, l'orientation de l'image est évaluée en comparant la largeur et la hauteur de l'image :

— Si largeur > hauteur, l'image est orientée horizontalement (mode paysage).

— Si hauteur > largeur, l'image est orientée verticalement (mode portrait).

— Si largeur == hauteur, l'image est carrée.

L'angle d'orientation est basé sur une formule dérivée de la géométrie :

$$\theta = \arctan\left(\frac{\text{opposé}}{\text{adjacent}}\right)$$

Si l'image est horizontalement alignée, le rapport $\frac{\text{hauteur}}{\text{largeur}}$ est utilisé. Si l'image est verticalement alignée, le rapport $\frac{\text{largeur}}{\text{hauteur}}$ est utilisé.

Ces calculs donnent l'angle θ en radians, qui est ensuite converti en degrés :

$$\text{angle (en degrés)} = \theta \times \frac{180}{\pi}$$

Utilisation pour Déterminer l'Orientation :

La valeur de l'angle peut donner une indication plus fine sur l'orientation de l'image :

— Si l'angle est proche de 0° ou 180°, l'image est presque parfaitement horizontale.

— Si l'angle est proche de 90° ou −90°, l'image est presque parfaitement verticale.

— Des angles intermédiaires (comme 45°) indiquent que l'image est inclinée.

8.1.1 Problème rencontré

Notre détection d'angle possède ainsi des limites, car, pour des images à 90° par exemple, l'image va être considérée comme étant droite. De plus, l'angle détectée possède des petites imprécisions, mais peu visibles.

8.2 Application en place de la fonction rotation

Après avoir obtenu l'angle pour lequel il faut tourner l'image, il nous suffit simplement d'appeler la fonction rotation, afin de la redresser de manière automatique.

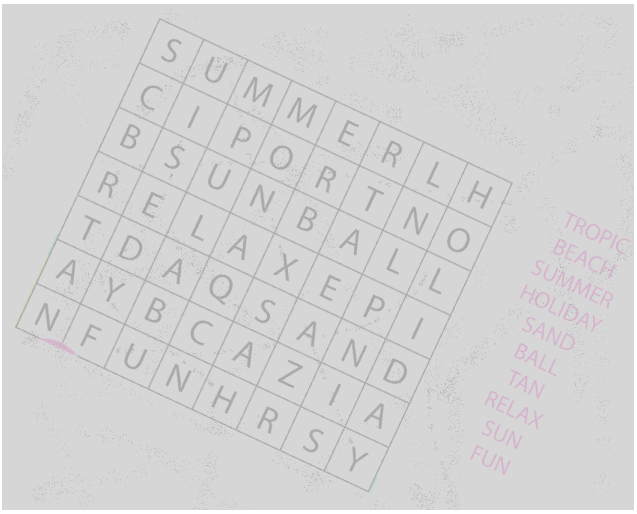


FIGURE 12 – Image avant redressement automatique

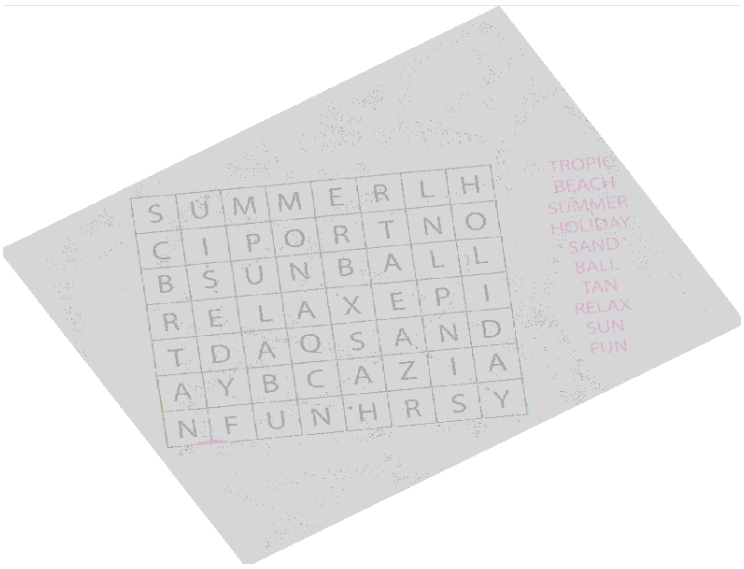


FIGURE 13 – Image après redressement automatique

9 Binarisation

Pour effectuer la binarisation de l'image, nous avons appliqué l'algorithme de Sauvola. Cet algorithme est basé sur la méthode de seuil local, qui adapte le seuil de binarisation en fonction des variations d'intensité de l'image. Il est particulièrement efficace pour les images ayant un contraste faible ou des variations d'éclairage.

La formule de Sauvola pour le calcul du seuil $T(x, y)$ au pixel (x, y) est donnée par :

$$T(x, y) = \mu(x, y) + k \cdot (\sigma(x, y) - m)$$

où :

- $\mu(x, y)$ est la moyenne des intensités des pixels dans un voisinage local autour du pixel (x, y).
- $\sigma(x, y)$ est l'écart type des intensités des pixels dans le même voisinage.
- k est un facteur de sensibilité qui peut être ajusté pour contrôler la binarisation.
- m est un paramètre constant (souvent pris égal à 128 pour des images en niveaux de gris).

Pour chaque pixel de l'image, nous comparons son intensité $I(x, y)$ avec le seuil calculé $T(x, y)$. Si $I(x, y) > T(x, y)$, le pixel est classé comme blanc (valeur 1), sinon il est classé comme noir (valeur 0). Cette approche permet de segmenter efficacement les objets d'intérêt dans l'image, en tenant compte des variations locales.

En utilisant cet algorithme, nous avons réussi à obtenir une binarisation robuste qui améliore la qualité des étapes suivantes dans le traitement de l'image.

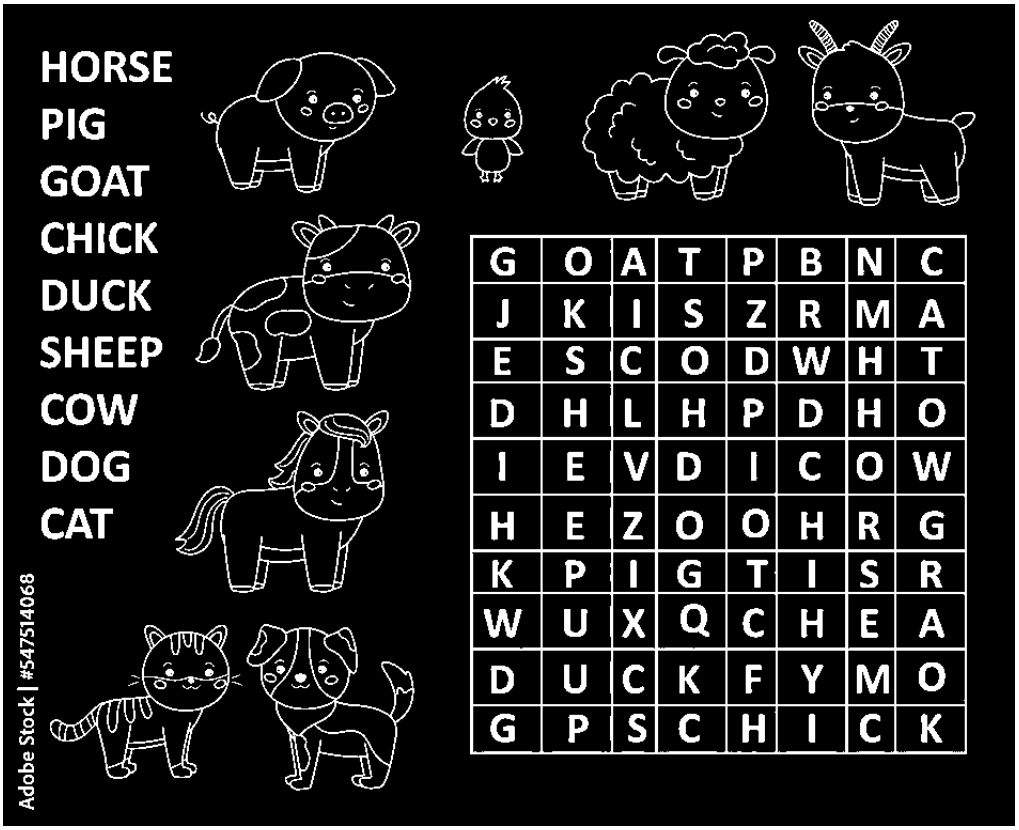


FIGURE 14 – Image après la binarisation

10 Détection de la Grille

Une fois l'image binarisée, nous allons commencer à détecter la grille et la liste de mots. Pour développer une stratégie efficace, nous avons étudié les méthodes de détection de grille et consulté de nombreuses ressources en ligne, notamment sur l'utilisation de Sobel et d'autres algorithmes de détection de grille.
Cependant, dans notre cas, il se peut qu'il n'y ait pas de grille sur l'image, et la liste de mots, quant à elle, n'est jamais encadrée par une grille. Nous avons donc choisi de procéder à l'inverse : au lieu de détecter la grille pour ensuite identifier les lettres, nous allons d'abord détecter les lettres, puis déterminer lesquelles appartiennent à la grille et lesquelles à la liste de mots.

Pour cela, nous appliquerons plusieurs étapes afin d'atteindre ce résultat.

10.1 Détection des formes

Nous commençons par détecter tous les amas de pixels blancs connectés entre eux, en utilisant un algorithme similaire à celui du "pot de peinture" des logiciels de dessin, qui parcourt l'image de manière récursive.

Deux matrices sont créées : l'une pour indiquer si un pixel a été visité et à quelle forme il appartient, et une autre qui stocke l'image sous forme de 1 (pixels noirs) et 0 (pixels blancs), afin de faciliter la distinction des pixels blancs et noirs dans la fonction récursive.

L'algorithme parcourt toute l'image et, dès qu'un pixel blanc non visité est trouvé, il initialise une nouvelle forme.
Structure de la forme :

```
typedef struct Shape
{
    int id; // Identifier of the shape
    int Cx, Cy; // Coordinates of the center of the
       shape
    int h, w; // Height and width of the shape

    // Coordinates of the edges of the shape
    int Maxj, Maxi;
    int Minj, Mini;

    int Matj, Mati; // Coordinates of the letter in
       the grid

    int Len; // Number of pixels in the shape
} Shape;
```

Un identifiant unique est attribué à chaque forme, puis l'algorithme parcourt récursivement les quatre pixels directement adjacents. Pour chaque pixel visité, sa valeur dans la matrice de vérification est remplacée par l'identifiant de la forme. Cette opération est répétée pour les pixels adjacents suivants.

Une fois la forme entièrement détectée, des informations comme le nombre de pixels constituant la forme ainsi que sa largeur et sa hauteur sont stockées.

Ensuite, un premier tri est effectué pour éliminer les formes trop grandes, par exemple celles qui occupent plus de la moitié de la taille de l'image.

Si une forme est valide, elle est ajoutée à la liste des formes. Pour cet algorithme, nous avons implémenté une structure de liste chaînée pour faciliter la manipulation des formes trouvées.

Structure de la liste chaînée :

```
typedef struct Node
{
    Shape* data;
    struct Node* next;
} Node;
```

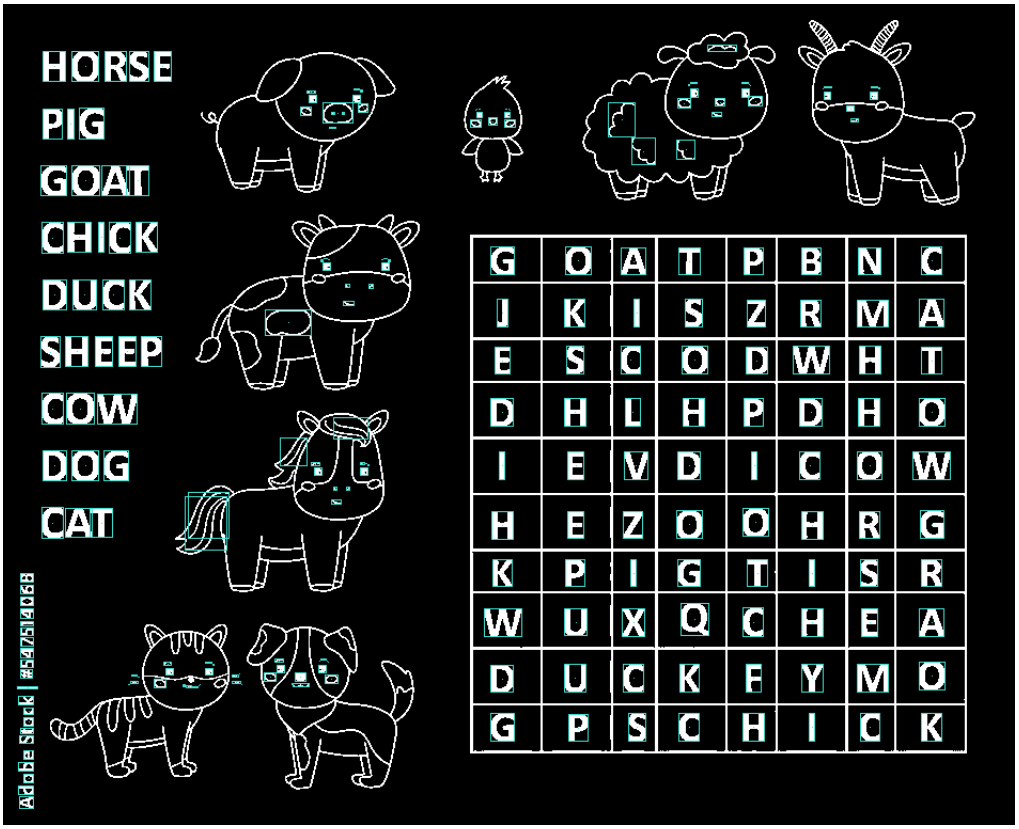


FIGURE 15 – Image après la détection des formes

10.2 Filtrage des formes

Une fois toutes les formes détectées, on filtre les formes non valides pour garder uniquement les lettres. Pour cela, nous calculons la somme de la largeur, de la hauteur et de la taille de toutes les formes, puis supprimons celles qui s'écartent trop de la moyenne. Ce filtre permet d'éliminer les formes trop petites.

En calculant la moyenne des hauteurs, on obtient la taille moyenne de la police, un paramètre efficace pour le filtrage.

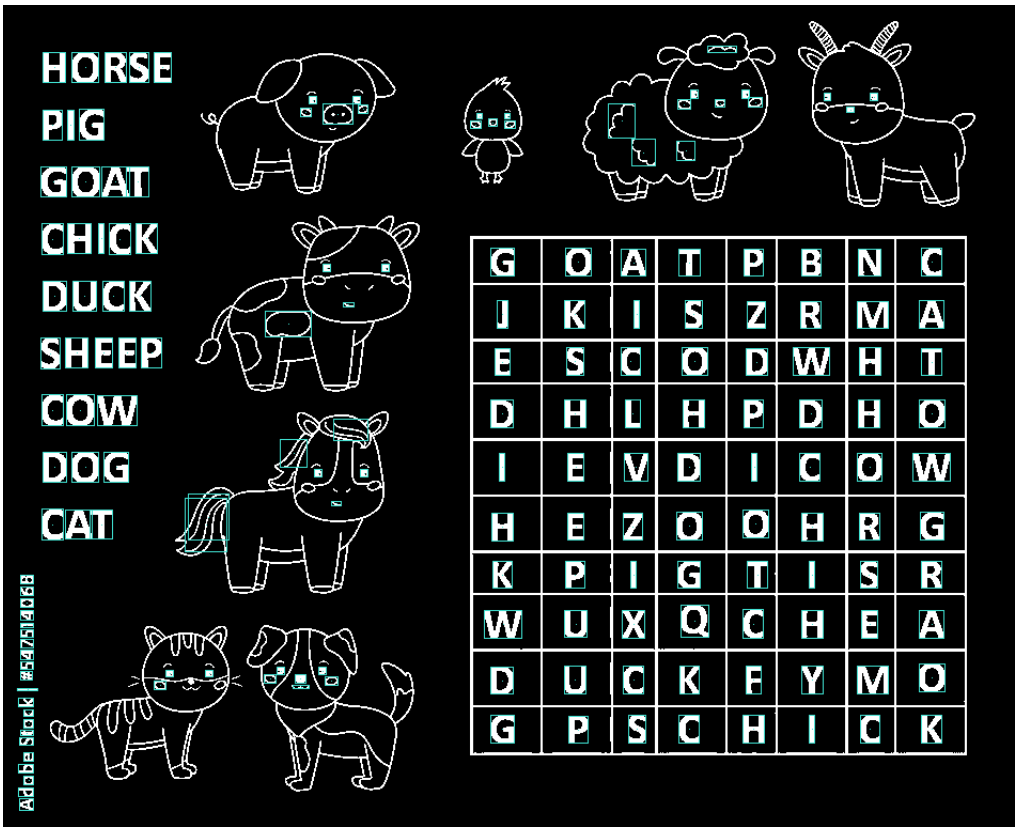


FIGURE 16 – Image après le filtrage des formes

10.3 Détection des groupes

À ce stade, la plupart des formes détectées correspondent à des lettres, mais certaines formes invalides peuvent subsister. Nous passons donc à la détection des groupes, en filtrant les groupes invalides.

L'objectif est d'extraire de l'image la grille de lettres et la liste de mots. Pour simplifier cette tâche, nous devons trouver des critères de détection communs à ces deux éléments. Nous allons considérer la grille de lettres comme une simple liste de mots parmi les autres. L'objectif est de détecter toutes les listes de mots présentes sur l'image, puis de distinguer celle qui correspond à la grille de lettres.

Pour détecter une liste de mots, nous commencerons par identifier toutes les lignes formées par des formes, que nous sauvegarderons sous forme de liste chaînée. Pour cela, nous parcourrons chaque forme non visitée et chercherons la forme la plus proche qui respecte trois critères : la forme testée doit être d'une taille similaire à celle de la forme actuelle, elle doit être suffisamment proche, et les deux formes doivent être alignées horizontalement. Une fois la forme ajoutée à la liste chaînée, nous recommençons la recherche à partir de cette nouvelle forme.

Cependant, cette méthode présente un problème. Prenons l'exemple suivant :

A B C D

Les lettres A, B, C et D sont alignées sur l'image. Si nous sommes sur la lettre B, imaginons que la forme la plus proche qui respecte tous les critères soit la lettre C. Nous poursuivrons donc la recherche à partir de la lettre C, puis la lettre D sera choisie ensuite. Cependant, la lettre A n'est jamais ajoutée à la ligne, et sera finalement ajoutée à une ligne où elle sera la seule forme.

Pour résoudre ce problème, si une forme qui respecte tous les critères appartient déjà à une ligne, nous fusionnerons les deux lignes.

Une fois toutes les lignes trouvées et sauvegardées dans un tableau, nous tenterons de former des groupes de lignes. Pour ce faire, nous comparerons chaque ligne avec toutes les autres. Nous chercherons les deux formes les plus proches entre deux lignes et vérifierons si elles sont suffisamment proches et si elles sont alignées verticalement. Si ces conditions sont remplies, nous regrouperons les lignes alignées verticalement dans un tableau.

À la fin de cette étape, nous aurons un tableau de listes de mots, soit un tableau de tableaux de lignes.

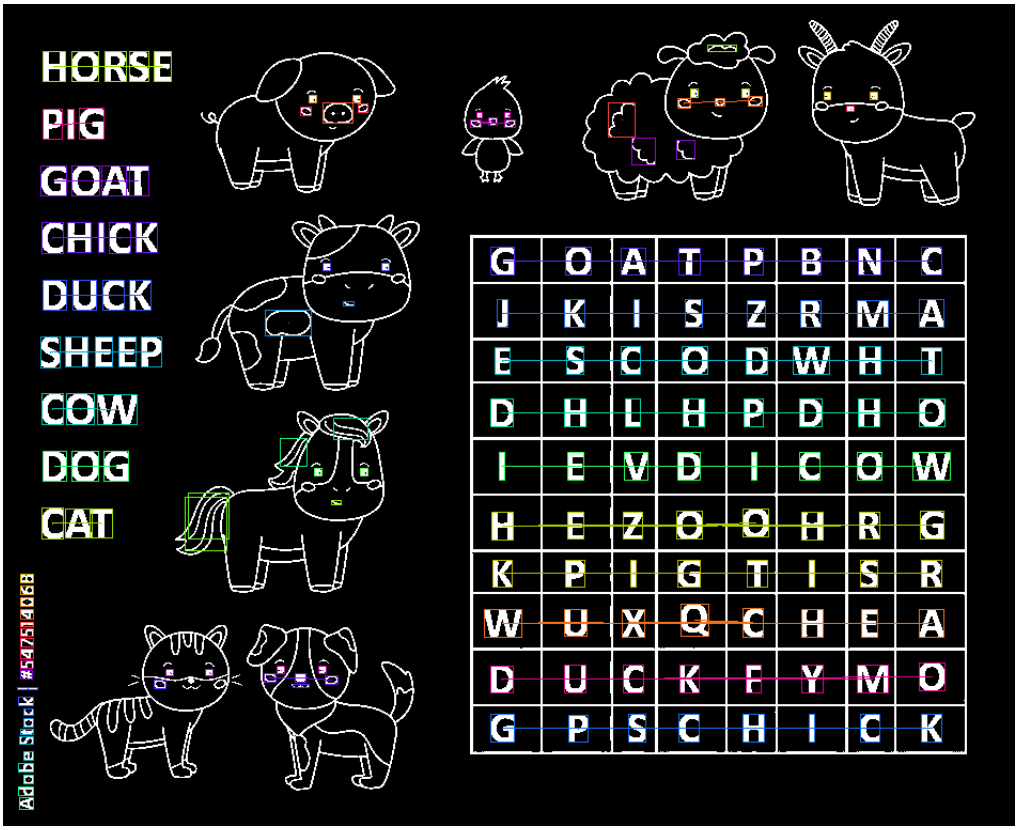


FIGURE 17 – Image après la détection des groupes

10.4 Filtrage des groupes

Une fois cette étape terminée, il nous reste à filtrer toutes les listes de mots invalides. Une liste de mots est considérée comme valide si elle contient plus d'un élément et que toutes les lignes qui la composent contiennent plus d'une forme. Nous procéderons également à d'autres vérifications, si nécessaire.

Nous obtenons alors plusieurs listes de lignes. Il est maintenant temps de distinguer la grille de lettres. Cela est relativement simple : il suffit de trouver la plus grande liste de lignes, dont toutes les lignes ont la même taille. Les autres listes de lignes seront considérées comme des listes de mots et seront regroupées en une seule liste.

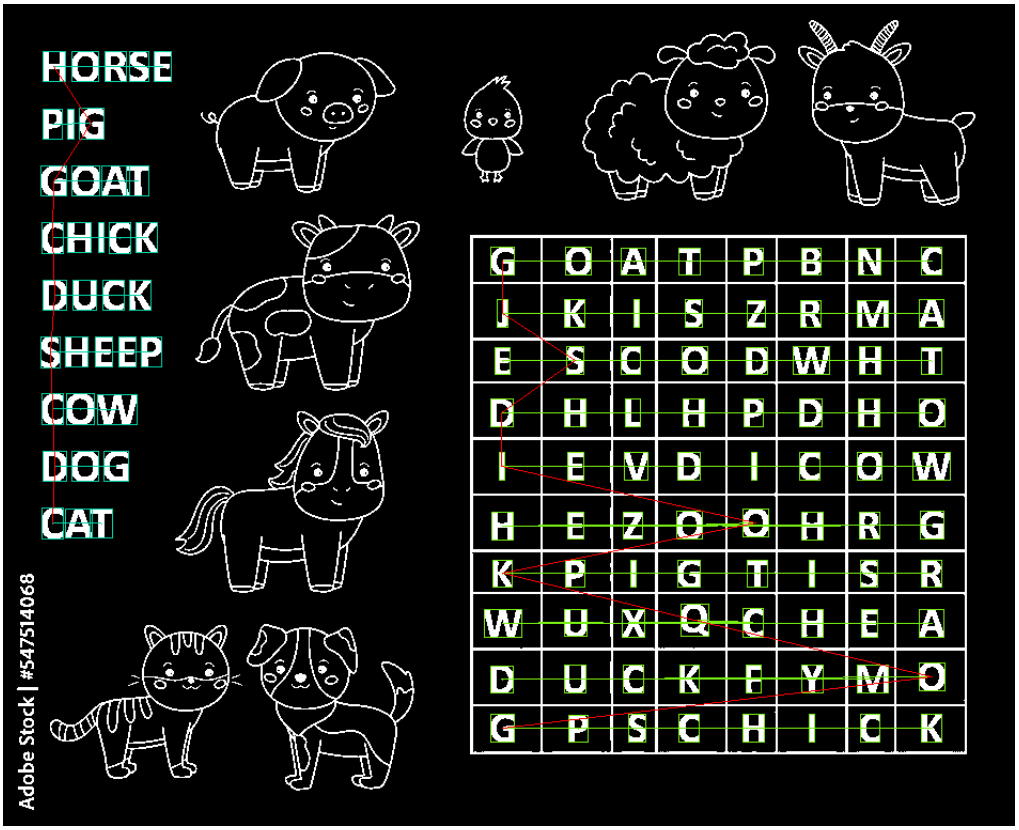


FIGURE 18 – Image après le filtrage des groupes

10.5 Triage des Clusters

Lorsque tous les filtres de Cluster et de Shape sont fini nous avons un Node*** qui comme expliqué précédemment contient des groupes de ligne. Le problème est que les lettres sont bien rangées par ligne mais pas dans le bon ordre. Par exemple, la première lettre n'est pas forcément le premier élément de la liste chaînée représentant la ligne. Nous appliquons un bubble sort sur les listes chaînées. Comme critère de trie nous prenons la coordonné x de la forme actuelle puis nous échangeons les données des nœuds de la liste chaînée.

Nous devons aussi mettre le cluster qui contient les lignes de la grille en première position. Pour cela nous parcourons tous les clusters calculons la longueur des listes chaînées. Le cluster dont toutes les lignes sont égales est la grille il nous suffit donc ensuite d'échanger les données sur lequel pointent les clusters.

10.6 Récupération des lettres

Nous disposons désormais d'une grille de formes et d'une liste de lignes, mais ces éléments seuls ne suffisent pas pour résoudre la grille. Nous allons donc transformer la grille en une matrice de lettres.

Une lettre est représentée par une structure contenant un caractère représentant la lettre, ainsi que les coordonnées x et y de la lettre sur l'image, afin de pouvoir dessiner la solution par la suite.

```
typedef struct Letter
{
    char letter;
    int x;
    int y;
} Letter;
```

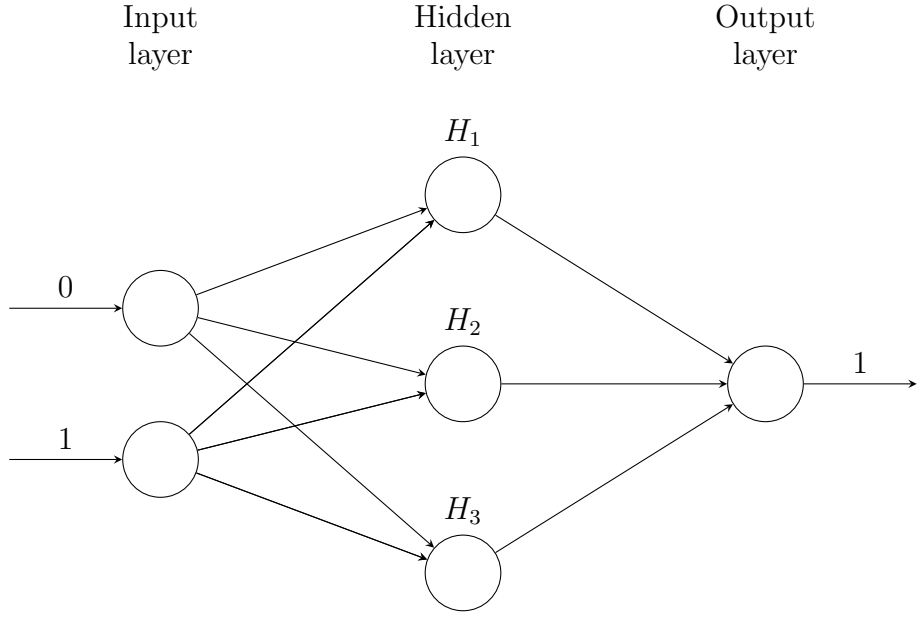
Pour déterminer quelle lettre correspond à quelle forme, nous ferons passer chaque forme dans un réseau de neurones, ce qui nous renverra la lettre associée. Nous en profiterons également pour écrire les lettres qui composent la grille dans un fichier, qui sera ensuite utilisé par le solveur.

Pour la liste de mots, nous appliquerons la même méthode, mais nous sauvegarderons également les mots sous forme de chaînes de caractères, afin de pouvoir les transmettre au solveur plus tard.

11 Réseau de neurones

Notre réseau de neurones suit une structure classique avec une couche d'entrée, une couche cachée et une couche de sortie. Le nombre de nœuds par couche varie selon l'application, comme le XOR ou le traitement d'image.

11.1 XOR



11.2 Traitement d'image

Le traitement d'image utilisé dans ce projet repose sur une structure similaire à celle employée dans l'algorithme XOR, mais avec une différence importante : nous avons 784 neurones d'entrée. Cette configuration correspond à chaque pixel d'une image de taille 28 par 28, qui est la dimension à laquelle les images sont redimensionnées avant d'être envoyées au réseau de neurones.

Ce redimensionnement permet de standardiser les images pour qu'elles puissent être traitées de manière uniforme par le réseau. Concernant les neurones cachés, nous avons opté pour un nombre de neurones supérieur de 1 au nombre de neurones d'entrée, car cette configuration a donné de bons résultats lors de nos tests. Nous avons également expérimenté avec des tailles de couches cachées correspondant à la moitié ou au double du nombre de neurones d'entrée, mais la configuration actuelle a montré les meilleurs résultats. Enfin, le nombre de neurones de sortie est fixé à 26, un pour chaque lettre de l'alphabet.

En ce qui concerne l'utilisation du réseau de neurones, le processus ne diffère pas beaucoup de celui utilisé pour l'algorithme XOR. Nous chargeons les données, nous faisons passer l'image que nous souhaitons tester à travers tous les neurones du réseau, et enfin, nous renvoyons le résultat, qui correspond à la lettre associée au neurone ayant la valeur la plus élevée dans la couche de sortie.

Cependant, ce qui change considérablement entre les deux projets, c'est l'entraînement du réseau de neurones.

Le premier défi majeur a été de trouver une base de données suffisamment grande pour entraîner le réseau de neurones de manière efficace. Une fois cette base de données identifiée, il a été nécessaire de prétraiter chaque image en la faisant passer par toutes les étapes du processus de filtrage des formes, afin que les données d'entraînement soient au même niveau que les formes qui seront ultérieurement envoyées au réseau de neurones dans le cadre de l'exécution du projet.

Une fois cette base de données préparée, nous avons commencé par faire passer les lettres de l'alphabet dans l'ordre alphabétique et ajuster les poids et les biais du réseau à chaque passage. Cependant, nous avons rapidement constaté que présenter les images dans le même ordre à chaque itération n'aidait pas le réseau à améliorer ses performances. En conséquence, nous avons décidé de mélanger l'ordre des lettres et des images lors de chaque passage sur la base de données. Cette méthode garantit que l'ordre des images change à chaque nouvelle itération, ce qui permet au réseau de s'entraîner de manière plus robuste.

Une fois ces ajustements effectués, nous avons lancé l'entraînement du réseau sur un serveur VPS hébergé chez l'un de nos membres. L'entraînement a duré une semaine complète.

Le réseau de neurones fonctionne très bien pour la plupart des lettres majuscules, mais il présente encore des difficultés avec certaines lettres comme "G" et "O", particulièrement lorsque ces lettres sont trop petites. De plus, le réseau ne parvient pas encore à reconnaître les lettres minuscules, ce qui constitue une limitation actuelle du modèle.

12 Solver

Pour résoudre le mot mêlé, on récupère la grille de lettres dans un fichier texte généré par le code. Cette grille est ensuite stockée dans une matrice de caractères.

Étape 1 Trouver la première lettre : on parcourt la grille jusqu'à la première lettre du mot.

Étape 2 Trouver la deuxième lettre : une fois la première lettre trouvée, on recherche la deuxième dans les 8 cases adjacentes. Si elle est trouvée, on détermine la direction pour le reste des lettres.

Étape 3 Trouver le reste du mot : en vérifiant que le mot peut tenir dans la direction sans dépasser les limites de la grille, on parcourt chaque lettre restante du mot.

Grâce à cet algorithme, on est certain de trouver le mot s'il existe dans la grille et d'en renvoyer les coordonnées. Ces informations seront très utiles plus tard pour indiquer visuellement l'emplacement de chaque mot dans la grille.

13 Affichage du résultat final

Une fois la grille et les listes de mots détectées, nous devons afficher la solution en traçant des lignes reliant les lettres des mots trouvés.

Pour afficher les solutions, nous avons implémenté une fonction capable de dessiner des lignes épaisses et colorées entre les lettres des mots détectés. Ces lignes représentent les mots trouvés et permettent de visualiser les solutions directement sur l'image originale. Les coordonnées du milieu de la première lettre et du milieu de la dernière sont envoyées à la fonction de traçage des lignes.

Chaque ligne est dessinée en respectant plusieurs paramètres :
- **Épaisseur** : La ligne a une épaisseur fixe, définie par un rayon en pixels.
- **Couleur** : Une couleur unique est attribuée à chaque mot pour faciliter la distinction.
- **Prolongation** : Les lignes sont étendues au-delà des premières et dernières lettres pour un affichage plus clair.

La fonction utilise un algorithme qui parcourt l'image pixel par pixel et modifie directement les couleurs des pixels selon le mot.

Voici le résultat final de notre programme après l'application de tous les algorithmes. Tous les mots sont trouvés, visualisés sous forme de lignes colorées.

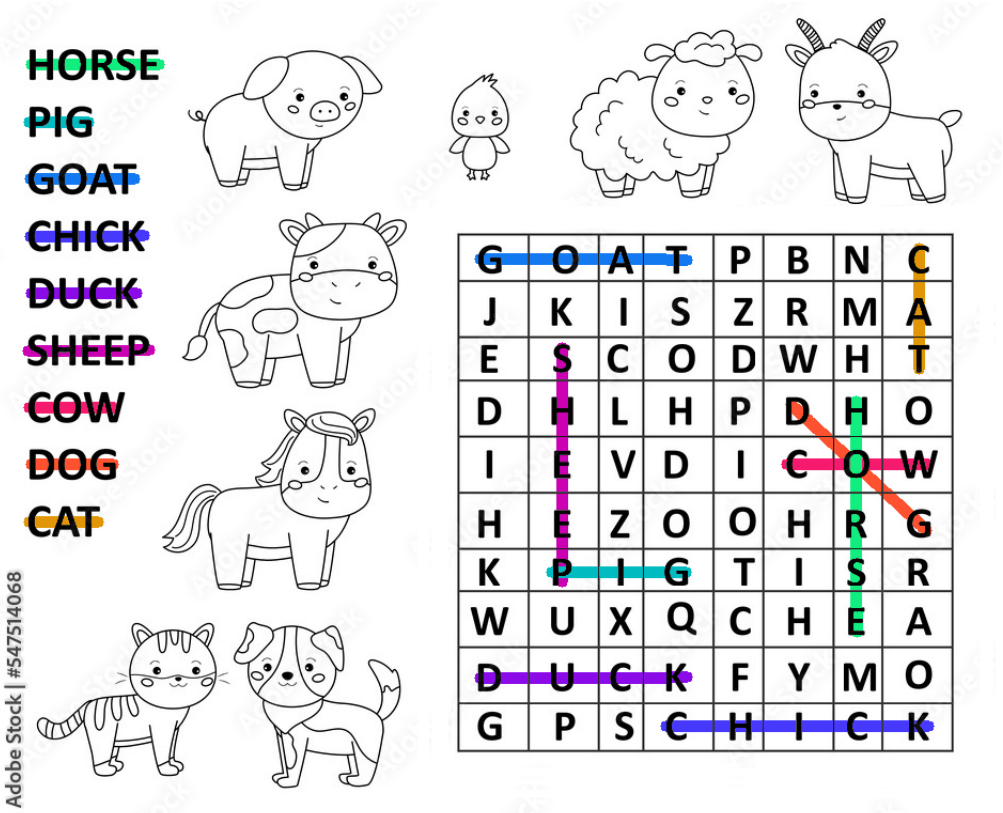


FIGURE 19 – Grille avec les mots trouvés visualisés

14 Interface

La création d'une interface cohérente et simple à utiliser est cruciale pour fluidifier l'expérience utilisateur. Elle a constitué une part importante du travail à effectuer sur le projet. L'interface est la liaison entre ce que voit l'utilisateur de notre logiciel et le traitement appliqué pour trouver la solution finale du problème initial.

14.1 Du concept...

La première étape a été de conceptualiser l'interface visuelle à l'aide du logiciel `Glade`. Notre objectif était de concevoir un ensemble simple avec uniquement des boutons à gauche pour modifier les paramètres de traitement, ainsi qu'une image à droite pour visualiser la solution et toutes les étapes qui ont mené à cette solution.

L'interface est donc composée d'une colonne de paramètres modifiables à gauche et d'une image dynamiquement modifiée à droite.

Dans la colonne de gauche, on retrouve différentes sections :
— Un titre et un logo créés par notre équipe, montrant à la fois deux mains qui s'entrechoquent pour le nom `CLAPS` ainsi qu'un petit robot analyseur pour symboliser la partie intelligence artificielle du projet.
— Un bouton pour importer depuis les fichiers de l'ordinateur une image à faire traiter par notre `OCR`. Celle-ci peut être modifiée à tout moment.
— Un curseur pour ajuster la rotation manuelle de l'image originale entre `-90` et `+90` degrés. Un bouton est également présent pour réinitialiser la valeur du curseur à `0`.
— Un bouton pour lancer le traitement de l'image. Celui-ci fait disparaître la section de rotation ainsi que sa propre section.
— Une série de boutons alternatifs pour sélectionner la visualisation de l'étape actuelle du traitement. Cela modifie dynamiquement l'image affichée à droite.
— Un bouton pour exporter l'image dans le format souhaité, à un emplacement sélectionné par l'utilisateur.

Dans la partie droite, on retrouve une image qui est modifiée en temps réel en fonction des actions de l'utilisateur. Par exemple, au lancement, il n'y a pas d'image, pendant la rotation, elle affiche l'image tournée, et lors de l'appui sur le bouton de traitement, elle affiche directement la solution du mot caché.

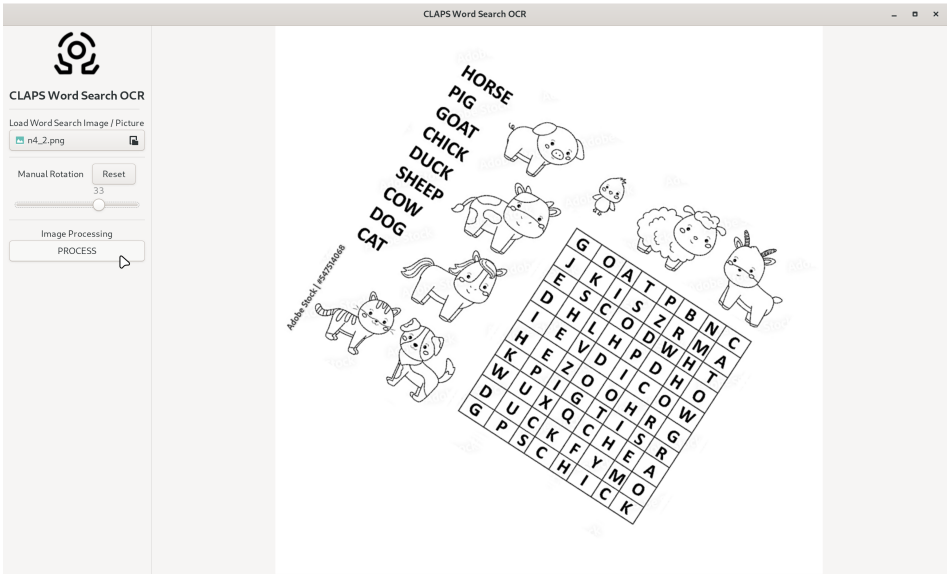


FIGURE 20 – Interface avant le processus de traitement

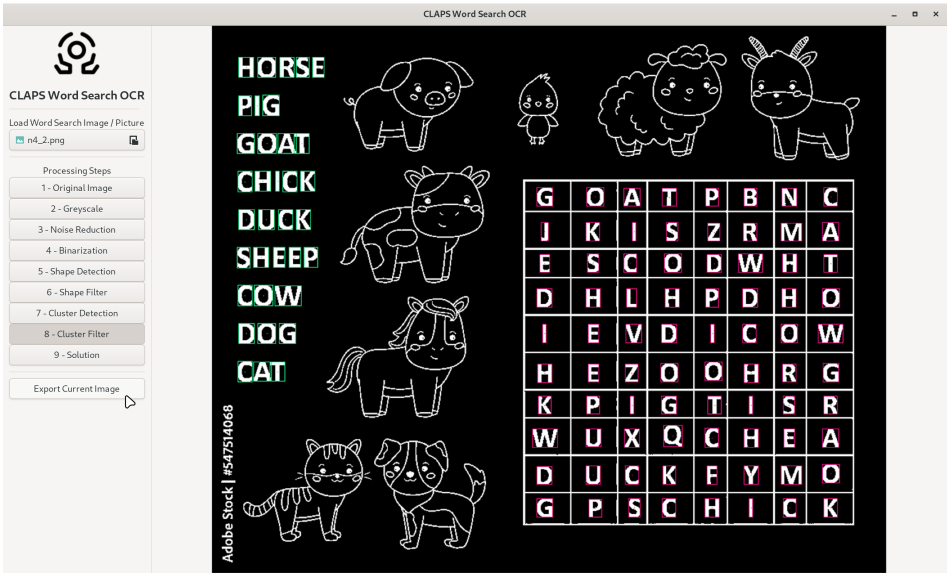


FIGURE 21 – Interface pendant le processus de traitement

14.2 ... Au concret

La deuxième étape a été de créer la liaison entre l'interface visuelle réalisée avec `Glade` et les fonctions codées dans le reste du projet, par l'intermédiaire de `GTK3`. `GTK` est par nature difficile à appréhender, car il est composé de dizaines de fonctions indépendantes du langage `C`.

Une fois la liaison créée, il a fallu faire fonctionner l'ensemble des éléments de l'interface.

L'image doit être facilement modifiable et doit conserver une taille adaptée à la hauteur de l'écran. Cela a été rendu possible grâce à `ImageMagick` et à son argument `-resize`, ainsi qu'à une implémentation dynamique de l'image.

Lors de l'importation d'une image, celle-ci doit être dupliquée et convertie en `.bmp` en deux exemplaires : l'un pour conserver la photo originale en cas d'exportation, et l'autre pour appliquer de potentielles rotations. Lors de l'appui sur le bouton de traitement, l'image doit être créée en 16 exemplaires différents : 8 pour l'exportation (l'image de l'étape `X` du traitement) en taille originale, et 8 autres pour l'affichage de l'image en taille redimensionnée, permettant ainsi de basculer rapidement entre les différents états une fois le traitement effectué, afin d'offrir l'expérience la plus fluide possible. Le traitement doit également créer un dossier d'images stockées sous le format `BMP` par cluster final, pour alimenter le réseau de neurones.

Le curseur de rotation, lors du relâchement de la souris après un appui, met à jour l'image stockée en lui appliquant la rotation du degré désiré. Lors de l'appui sur le bouton de réinitialisation, la valeur du curseur est remise à `0`, et l'image est remplacée par l'image originale.

Le bouton de traitement masque la section de rotation ainsi que sa propre section, pour laisser place aux boutons d'étapes et à la section d'exportation. Cependant, il conserve le bouton d'importation, qui reste accessible à tout moment pour changer l'image d'entrée, ce qui a pour effet de réinitialiser la visualisation des boutons.

Lorsqu'un bouton de sélection d'étape est activé, il met à jour l'image à droite et désélectionne le bouton actuellement activé, agissant ainsi comme un bouton alternatif parmi les autres boutons de sélection d'étapes.

Lors de l'appui sur le bouton d'exportation, une fenêtre de dialogue s'ouvre et permet de choisir un emplacement sur l'ordinateur pour sauvegarder l'image actuellement affichée dans son format original. Le nom par défaut est `solution.png`, mais il peut être modifié, ainsi que le format, grâce à l'outil `ImageMagick`, qui convertit automatiquement l'image dans le format souhaité en fonction de l'état actuel du traitement (`CurrentState`).

À la fermeture du programme ou lors de l'importation d'une nouvelle image, le logiciel supprime toutes les images temporaires présentes dans `/output/` ainsi que toutes les images de lettres présentes dans les sous-dossiers des clusters. Il termine enfin en fermant proprement l'interface `GTK`.

15 Conclusion

Nous avons pu réaliser l'ensemble de nos objectifs associés à ce projet :
- 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 (Solver)
- Affichage de la résolution sous forme de lignes
- Réseau de neurones fonctionnel

Pour ce deuxième rapport, nous avons réalisé les objectifs suivants :
- Réduction de bruit
- Rotation automatique
- Réseau de neurones

Ce dernier rapport marque la fin de notre projet, bien qu'il reste des points à améliorer. L'aventure réalisée pour ce projet nous a appris beaucoup sur la création d'interfaces, les réseaux de neurones et le traitement d'image.