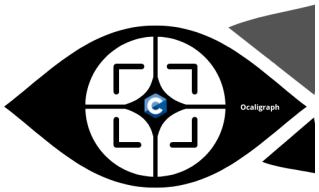


OCR RAPPORT DU 3 NOVEMBRE

ÉCOLE POUR L'INFORMATIQUE ET LES TECHNIQUES AVANCÉES

GROUPE OCALIGRAPH

HTTPS://WWW.OCALIGRAPH.COM/

À EPITA

PROMO 2023-2028

GAETAN SUILLEROT, KEANY VY KHUN, THEO BRULIER, NATHAN GILLOTIN

L'ÉQUIPE OCALIGRAPH

LE 03/11/2024

V1.0 - Prod

Résumé

Le projet OCaligraph est un logiciel de reconnaissance optique de caractères (OCR) développé par des étudiants de deuxième année à l'EPITA.

Ce document présente l'état d'avancement du projet pour la première soutenance de novembre.

À ce stade, l'équipe a réussi à mettre en place plusieurs éléments clés, notamment un solveur pour identifier les mots dans la grille de mots croisés, le prétraitement des images à l'aide de divers filtres pour améliorer leur lisibilité, ainsi que le découpage des images pour extraire les éléments pertinents.

De plus, une fonction de rotation d'image a été intégrée pour garantir une orientation correcte de la grille.

Le rapport décrit les techniques utilisées, telles que le filtre de Canny et la transformation de Hough pour la détection des bords et des lignes, tout en abordant les défis rencontrés durant le développement.

Enfin, il souligne les objectifs futurs, notamment l'amélioration du prétraitement et l'optimisation des fonctions existantes, offrant ainsi un aperçu complet des avancées du projet, des aspects techniques et organisationnels de l'équipe.

Mots clefs

Filtre de Canny, Transformation de Hough, Masque gaussien, Gradient d'intensité, Suppression des non-maxima, Seuillage adaptatif, Filtre grayscale, Filtre médian, Filtre moyen, Bibliothèque SDL, Convolution, Détection des bords, Segmentation, Prétraitement d'image, Rotation d'image, Découpage d'image, Solveur de mots croisés, Réseau neuronal

Table des matières

Chapitre 1

L'équipe du projet

— Gaetan Suillerot : Mon intérêt pour l'informatique a commencé au collège avec Scratch. Au lycée, la création de projets en équipe est devenue une passion. J'ai choisi l'EPITA Paris pour sa réputation internationale. Mon rôle de chef de projet me tient à cœur, car la réussite passe par une bonne communication et cohésion d'équipe. Ce projet me fera découvrir de nouveaux domaines et enrichir mes compétences.

— Keany Vy Khun : Fervent défenseur de l'open-source depuis 2019, j'ai contribué au code source de Dogecoin Core et je participe activement à des repair cafés. J'ai commencé l'informatique en 6ème en mettant en place des nœuds de sortie Tor par la configuration du torrc et depuis j'ai pu cultiver une expérience en cybersécurité que je souhaite approfondir dans un cadre plus formel.

— Theo Brulier : Ma passion pour l'informatique a commencé au collège, lorsque j'ai découvert la programmation avec Arduino, ce qui m'a permis de réaliser mes premiers projets. Aujourd'hui, en deuxième année d'ingénierie en informatique, je suis déterminé à enrichir mes compétences et à approfondir mes connaissances. Ce projet présente des applications concrètes dans de nombreux domaines et m'encourage à m'investir pleinement.

— Nathan Gillotin : Étudiant en deuxième année à l'EPITA, j'ai rejoint ce groupe aléatoirement. Nous avons appris à nous connaître et à structurer rapidement notre travail, développant ainsi des compétences en communication et gestion de projet. Le traitement d'image m'intéresse particulièrement pour ses applications variées en médecine, automobile et loisirs.

Chapitre 2

Répartition des tâches

Rôle conception	Membre principal	Membre secondaire
Gestion de l'image :		
Prétraitement	Nathan Gillotin	Gaetan Suillerot
Rotation de l'image	Nathan Gillotin	Gaetan Suillerot
Segmentation	Theo Brulier	NA
Transformée de Hough	Theo Brulier	NA
Détection des contours	Theo Brulier	NA
Détection du carré	Theo Brulier	Keany Vy Khun
Reconnaissance des caractères :		
Réseau de neurones	Keany Vy Khun	Theo Brulier
Banque d'images	Theo Brulier	Keany Vy Khun
Résolution de la grille	Gaetan Suillerot	NA
Autres :		
Parser terminal	Keany Vy Khun	NA
Interface CLI	Keany Vy Khun	NA
Site Web	Keany Vy Khun	NA
Documentation	Keany Vy Khun	NA

Table 1 - Tableau de répartition des tâches

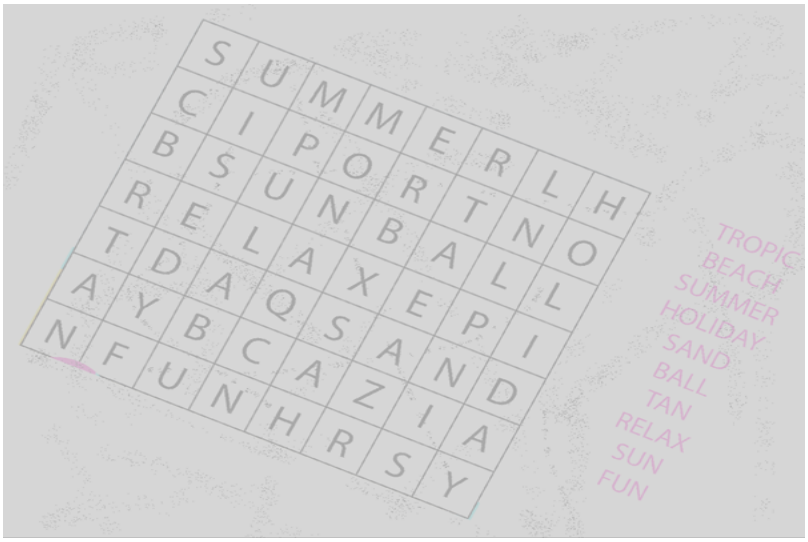
Chapitre 3

Techniques de prétraitement d'images

Le prétraitement de l'image est une étape cruciale pour faciliter la détection ultérieure. Cette phase consiste à supprimer les couleurs de l'image et à accentuer les traits pour les rendre plus lisibles par la machine. Pour ce faire, nous appliquons plusieurs filtres successivement sur l'image, en utilisant la bibliothèque SDL2.

SDL2 est une bibliothèque en C utilisée pour la gestion des graphismes, de l'audio, des entrées, et d'autres fonctionnalités liées aux applications multimédia et aux jeux vidéo.

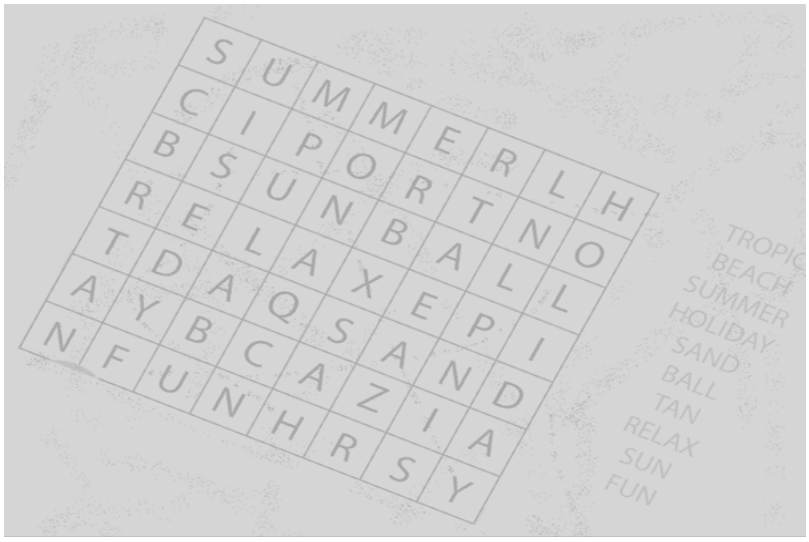
Voici les étapes du prétraitement, illustrées par une image difficile à traiter en raison de ses traits peu marqués :



3.0.1 Conversion en niveaux de gris (Grayscale)

Nous utilisons la fonction grayscale, implémentée lors du premier TP de l'année. Cette fonction modifie chaque pixel de l'image selon la formule :

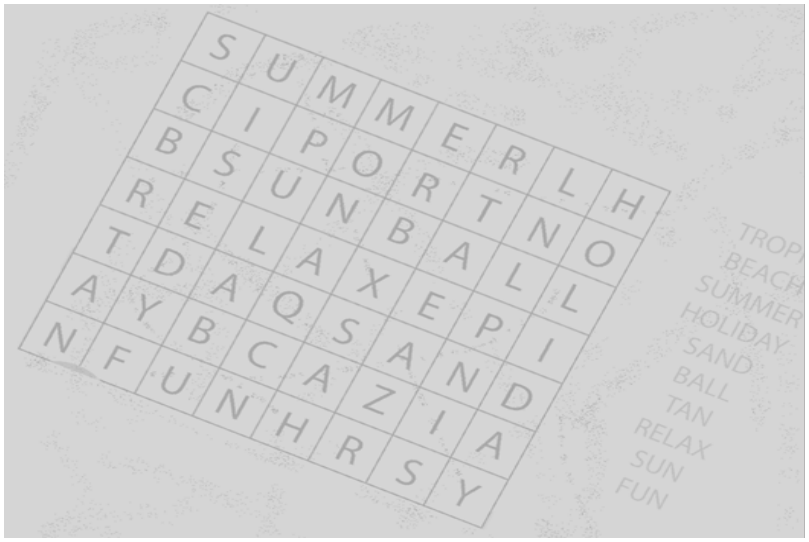
$$\text{pixel} = 0.3r + 0.59g + 0.11b$$



Cette opération transforme l'image en niveaux de gris.

3.0.2 Filtre Médian

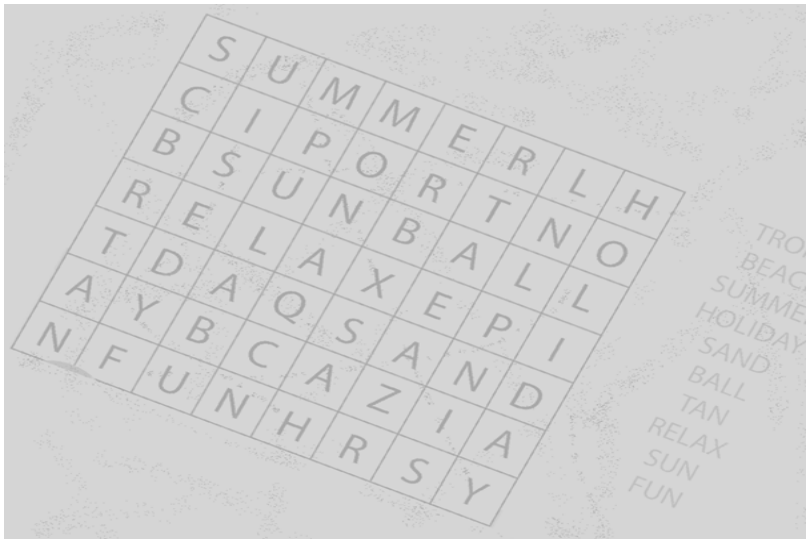
Ce filtre parcourt chaque pixel de l'image. Pour chacun, il crée une liste contenant le pixel et ses 8 voisins adjacents. Le pixel est ensuite remplacé par la valeur médiane de cette liste (l'élément d'indice 4 une fois la liste triée).



3.0.3 Filtre Moyen

Similaire au filtre médian, mais le pixel est remplacé par la moyenne des valeurs de la liste au lieu de la médiane.

Les filtres médian et moyen ont des effets subtils à l'œil nu, mais sont essentiels pour le bon fonctionnement des filtres suivants.

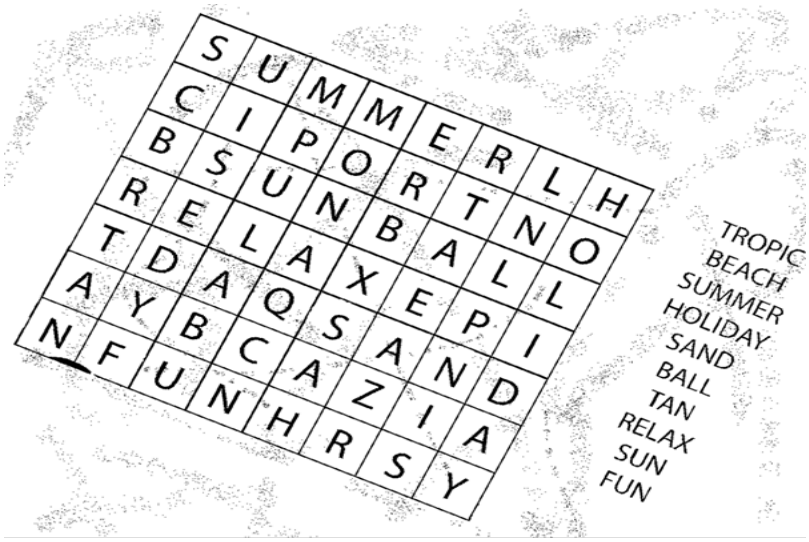


3.0.4 Seuillage Adaptatif

L'image est divisée en sous-parties, et un seuil est calculé pour chacune. Si la valeur d'un pixel est supérieure au seuil, il devient blanc ; sinon, il devient noir. La taille des sous-parties est déterminée par le niveau de bruit de l'image, calculé par la fonction "noiseLevel".

La fonction noiseLevel parcourt chaque pixel de l'image, calcule la moyenne du pixel et de ses voisins, puis applique la formule :

$$valeur = 1 - \frac{pixel}{moyenne}$$



Si cette valeur dépasse un seuil fixé à 0.5, le pixel est considéré comme du bruit.

Points d'amélioration :

- Perfectionner les filtres existants
- Ajouter de nouveaux filtres (ex : filtre de Gauss, filtre de contraste)
- Optimiser les fonctions pour accélérer le processus

Difficultés rencontrées :

- Documentation : trouver les méthodes les plus adaptées parmi les nombreuses options disponibles
- Erreurs de segmentation : identification difficile de l'origine du problème

Rotation d'image :

La rotation d'image permet de positionner correctement la grille de mots croisés. Utilisant la bibliothèque SDL, nous chargeons l'image dans une texture, appliquons la rotation, puis transformons le résultat en surface pour le sauvegarder.

Chapitre 4

Architecture et fonctions du solveur

Le solveur est composé de 6 fonctions principales. Son objectif est de trouver un mot spécifique dans une grille de mots croisés stockée dans un fichier texte. Pour accomplir cette tâche, plusieurs paramètres sont pris en compte et traités par différentes fonctions.

Initialisation et structures de données :

- Des constantes sont définies pour la taille maximale de la grille, la longueur maximale d'un mot, et le nombre maximal de positions de départ possibles.
- Les variables globales sont initialisées, notamment "wordsearch", une matrice 2D représentant la grille de recherche.
- Une structure "position", est créée pour stocker les coordonnées x et y de chaque mot dans la grille.

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

// Define constants for maximum sizes
#define MAX_GRID_SIZE 100 // Maximum size of the wordsearch grid
#define MAX_WORD_LENGTH 50 // Maximum length of a word to search
#define MAX_POSITIONS 1000 // Maximum number of starting positions for a word

// Structure to represent a position in the grid
typedef struct {
    int x; // Row index
    int y; // Column index
} Position;

// Global variables
char wordsearch[MAX_GRID_SIZE][MAX_GRID_SIZE]; // 2D array to store the wordsearch grid
int rows, cols; // Dimensions of the wordsearch grid
```

Fonctions principales :

1. is-valid-index : Cette fonction vérifie si une position donnée (line-num, col-num) est valide dans la grille.

```c
/**
 * Check if a given position is within the grid boundaries
 * line_num Row index
 * col_num Column index
 * return 1 if valid, 0 if not
 */

int is_valid_index(int line_num, int col_num) {
  return (line_num >= 0 && line_num < rows && col_num >= 0 && col_num < cols);
}
```

2. chars-match : Cette fonction vérifie si les caractères trouvés dans la grille correspondent au mot recherché.

```c
/**
 * Check if the characters found match the beginning of the word
 * found Array of characters found in the grid
 * word Word being searched for
 * found_len Length of the found characters
 * return 1 if match, 0 if not
 */

int chars_match(const char* found, const char* word, int found_len) {
  for (int i = 0; i < found_len; i++) {
    if (found[i] != word[i]) {
      return 0;
    }
  }
  return 1;
}
```

3. print-coord : Affiche les coordonnées finales (première et dernière lettre) en fonction de la direction du mot trouvé.

```c
/**
 * Print coordinates depending on the directions dx and dy
 * sx Start X
 * sy Start Y
 * ex End X
 * ey End Y
 * dx X direction (-1, 0, or 1)
 * dy Y direction (-1, 0, or 1)
 */

void print_coord(int sx, int sy, int ex, int ey, int dx, int dy) {
  if (dx == -1 && dy == 1) {
```

```
    printf("(%i,%i)(%i,%i)\n",ey,ex,sy,sx);
  }
  if (dx == 0 && dy == 1) {
    printf("(%i,%i)(%i,%i)\n",sy,sx,ey,ex);
  }
  if (dx == 1 && dy == 1) {
    printf("(%i,%i)(%i,%i)\n",sy,sx,ey,ex);
  }
  if (dx == -1 && dy == 0) {
    printf("(%i,%i)(%i,%i)\n",ey,ex,sy,sx);
  }
  if (dx == 1 && dy == 0) {
    printf("(%i,%i)(%i,%i)\n",sy,sx,ey,ex);
  }
  if (dx == -1 && dy == -1) {
    printf("(%i,%i)(%i,%i)\n",ey,ex,sy,sx);
  }
  if (dx == 0 && dy == -1) {
    printf("(%i,%i)(%i,%i)\n",ey,ex,sy,sx);
  }
  if (dx == 1 && dy == -1) {
    printf("(%i,%i)(%i,%i)\n",sy,sx,ey,ex);
  }
}
```

4. check-dir : Pour un mot donné, cette fonction vérifie s'il existe à partir d'un point de départ spécifié dans une direction donnée. Elle utilise :

   - Une liste "found-char" pour stocker les caractères rencontrés dans la direction (dx, dy).
   - Une liste "final" pour enregistrer les coordonnées de chaque caractère trouvé. La fonction appelle ensuite print-coord pour afficher les résultats.

```
/**
 * Check if the word exists in a specific direction from a starting position
 * word Word to search for
 * start_pos Starting position
 * dx X direction (-1, 0, or 1)
 * dy Y direction (-1, 0, or 1)
 * return 1 if word found, 0 if not
 */

int check_dir(const char* word, Position start_pos, int dx, int dy) {
  char found_chars[MAX_WORD_LENGTH];
  Position current_pos = start_pos;
  Position positions[MAX_WORD_LENGTH];
  int pos_count = 0;
  int word_len = strlen(word);
  int final[2 * MAX_WORD_LENGTH];
  int index = 0;

  found_chars[0] = wordsearch[start_pos.x][start_pos.y];
```

```
  positions[pos_count] = start_pos;
  pos_count++;

  for (int i = 1; i < word_len; i++) {
    current_pos.x += dx;
    current_pos.y += dy;
    if (!is_valid_index(current_pos.x, current_pos.y)) {
      return 0;
    }

    found_chars[i] = wordsearch[current_pos.x][current_pos.y];
    positions[pos_count] = current_pos;
    pos_count++;

    if (!chars_match(found_chars, word, i + 1)) {
      return 0;
    }
  }

  // Word found, print the coordinates
  for (int x = 0; x < rows; x++) {
    for (int y = 0; y < cols; y++) {
      for (int z = 0; z < pos_count; z++) {
        if (positions[z].x == x && positions[z].y == y) {
          final[index] = x;
          index++;
          final[index] = y;
          index++;
        }
      }
    }
  }

  print_coord(final[0],final[1],final[index - 2],final[index - 1],dx,dy);
  return 1;
}
```

5. check-start : Appelle check-dir pour chaque direction possible à partir d'un point de départ. Retourne 1 si le mot est trouvé, 0 sinon.

```
/**
 * Check all 8 directions from a starting position for the word
 * word Word to search for
 * start_pos Starting position
 * return 1 if word found in any direction, 0 if not
 */

int check_start(const char* word, Position start_pos) {
  int directions[8][2] = {{-1,1}, {0,1}, {1,1}, {-1,0}, {1,0}, {-1,-1}, {0,-1},
      {1,-1}};
  for (int i = 0; i < 8; i++) {
    if (check_dir(word, start_pos, directions[i][0], directions[i][1])) {
      return 1;
```

```
    }
  }
  return 0;
}
```

6. find-word : Fonction principale de recherche. Elle parcourt la grille pour trouver le premier caractère du mot, puis :
    - Attribue les coordonnées correspondantes à la structure "Position".
    - Appelle check-start pour toutes les positions de départ potentielles.
    - Affiche les coordonnées si le mot est trouvé, ou "Not found" dans le cas contraire.

```
/**
 * Find a word in the wordsearch grid
 * word Word to search for
 */

void find_word(const char* word) {
  Position start_pos[MAX_POSITIONS];
  int start_pos_count = 0;
  char first_char = word[0];

  // Find all positions of the first character of the word
  for (int i = 0; i < rows; i++) {
    for (int j = 0; j < cols; j++) {
      if (wordsearch[i][j] == first_char) {
        start_pos[start_pos_count].x = i;
        start_pos[start_pos_count].y = j;
        start_pos_count++;
      }
    }
  }

  // Check each starting position for the word
  for (int i = 0; i < start_pos_count; i++) {
    if (check_start(word, start_pos[i])) {
      return;
    }
  }
  printf("Not found\n");
}
```

6. La fonction principale du solveur :
    - Ouvre le fichier contenant la grille (en mode lecture).
    - Convertit tous les caractères (du fichier et du mot recherché) en majuscules pour éviter les problèmes de casse.
    - Appelle find-word pour effectuer la recherche.

```c
int main(int argc, char* argv[]) {

  if (argc != 3) {
    printf("USAGE : ./solver FILE WORD\n");
    return 0;
  }
  char *filename;
  char *word;
  char line[MAX_GRID_SIZE];
  // Get the filename from the user
  filename = argv[1];
  // Open and read the file
  FILE *file = fopen(filename, "r");
  if (file == NULL) {
    printf("Unable to open file.\n");
    return 1;
  }

  // Read the wordsearch grid from the file
  rows = 0;
  while (fgets(line, sizeof(line), file) && rows < MAX_GRID_SIZE) {
    cols = strlen(line) - 1; // -1 to remove newline
    for (int i = 0; i < cols; i++) {
      wordsearch[rows][i] = toupper(line[i]);
    }
    rows++;
  }

  fclose(file);

  // Get the word to search
  word = argv[2];

  // Convert word to uppercase for case-insensitive search
  for (int i = 0; word[i]; i++) {
    word[i] = toupper(word[i]);
  }

  find_word(word);
  return 0;
}
```

Chapitre 5

Méthodes de détection et extraction d'éléments

5.0.1 I. Détection

Le mode verbose (-v) a été intégré dès le début, facilitant le débogage en fournissant des informations détaillées sur les opérations effectuées. Cette fonctionnalité s'est avérée particulièrement utile dans cette partie complexe.

La détection des éléments d'intérêt dans l'image de mots mêlés se déroule en plusieurs étapes, utilisant principalement le filtre de Canny et la transformation de Hough.

5.0.2 Détection des bords

Pour la détection des bords de l'image, nous avons opté pour le filtre de Canny, qui offre une identification précise des contours avec une complexité raisonnable. Ce filtre comprend plusieurs étapes :

5.0.3 Création automatisée d'un masque gaussien

Une fonction génère un masque gaussien normalisé de taille et de force (écart-type σ) ajustables, offrant une flexibilité dans le code. Chaque terme du masque est calculé selon la formule :

$$G(x, y) = \frac{1}{2\pi\sigma^2} \cdot e^{-\frac{x^2+y^2}{2\sigma^2}}$$

5.0.4 Application du filtre associé au masque gaussien

La convolution est appliquée à tous les pixels de l'image. Pour chaque pixel $I(x, y)$, la convolution avec le masque gaussien G est effectuée :

$$I'(x, y) = \sum_{m=-k}^{k} \sum_{n=-k}^{n} G(m, n) \cdot I(x + m, y + n)$$

Cette étape réduit le bruit et prépare l'image pour les étapes suivantes.

5.0.5 Calcul des gradients d'intensité et des orientations des contours

Pour ce faire, nous commençons par créer un gradient G_x et un gradient G_y à l'aide d'un filtre de Sobel. Nous utilisons un masque 5x5 pour une précision accrue par rapport au masque 3x3 habituellement utilisé.

Sobel 3x3 :

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * \text{Image}$$

$$G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \text{Image}$$

Sobel 5x5 :

$$G_x = \begin{bmatrix} -2 & -1 & 0 & 1 & 2 \\ -4 & -2 & 0 & 2 & 4 \\ -6 & -3 & 0 & 3 & 6 \\ -4 & -2 & 0 & 2 & 4 \\ -2 & -1 & 0 & 1 & 2 \end{bmatrix} * \text{Image}$$

$$G_y = \begin{bmatrix} 2 & 4 & 6 & 4 & 2 \\ 1 & 2 & 3 & 2 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ -1 & -2 & -3 & -2 & -1 \\ -2 & -4 & -6 & -4 & -2 \end{bmatrix} * \text{Image}$$

La magnitude du gradient est calculée par :

$$G = \sqrt{(G_x)^2 + (G_y)^2}$$

L'orientation des contours est déterminée par :

$$\text{atan2}(G_x, G_y)$$

arrondie à l'angle le plus proche parmi $\{0°, 45°, 90°, 135°\}$.

5.0.6 Suppression des non-maxima

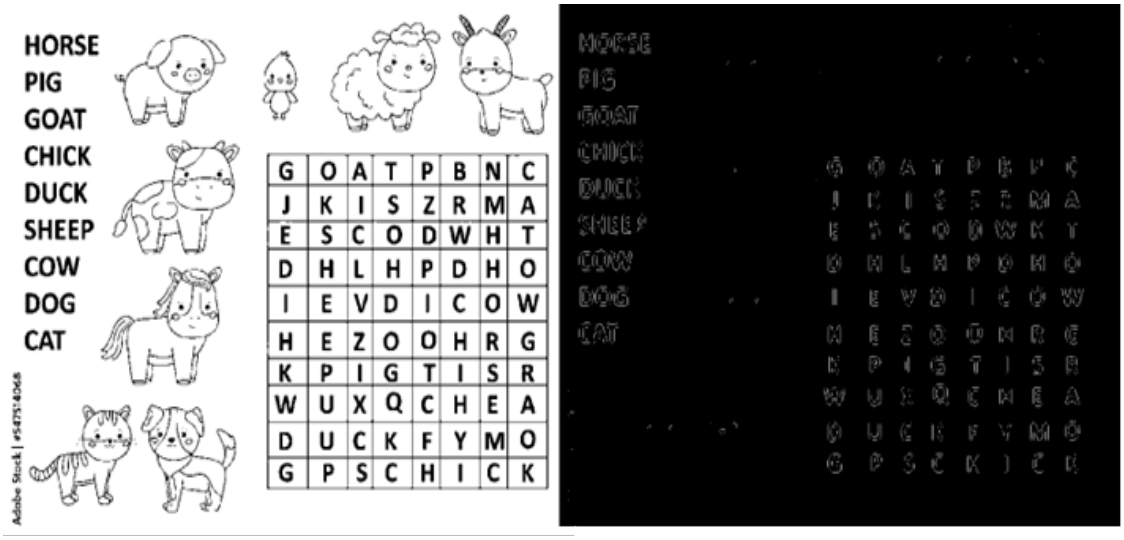
Cette étape élimine les pixels ne correspondant pas à des bords réels en ne conservant que ceux ayant une magnitude de gradient maximale dans leur direction.

5.0.7 Seuils de contours

Deux seuils (élevé et faible) sont appliqués pour classer les pixels comme bords. Les pixels entre les deux seuils ne sont conservés que s'ils sont connectés à des bords forts.

5.0.8 Sauvegarde de l'image obtenue

L'image résultante après l'application du filtre Canny est sauvegardée pour être utilisée lors de la transformation de Hough.

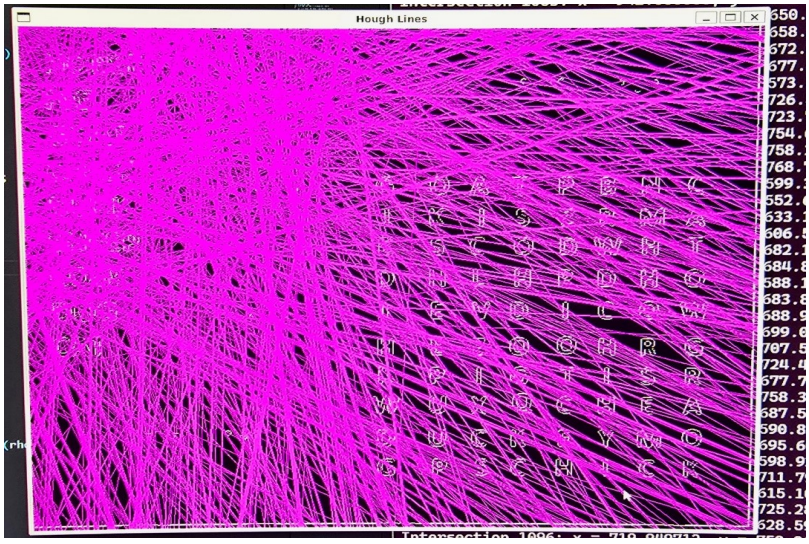


5.0.9 Détection des lignes

La transformation de Hough est utilisée pour identifier les lignes présentes dans l'image après la détection des bords. Elle transforme les points de l'espace image en une représentation paramétrique :

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

où ρ est la distance à l'origine, et θ est l'angle de la ligne. Malheureusement, la transformation de Hough n'est pas encore entièrement fonctionnelle dans notre implémentation.



Après la détection des lignes, nous déterminons les coordonnées de la grille et de la liste de mots en identifiant les intersections des lignes détectées. Cela permet de localiser les cellules de la grille et d'en extraire les lettres qui la composent, ainsi que d'identifier la liste de mots pour en extraire les mots et les lettres correspondantes. Ces coordonnées seront ensuite utilisées pour découper la grille et sauvegarder les différents éléments.

Cependant, comme nous n'avons pas réussi à implémenter la transformation de Hough, cette partie n'a pas pu être réalisée. Néanmoins, nous avons trouvé une autre méthode pour

obtenir les coordonnées de la grille et des lettres qui la composent. Voici les étapes de cette méthode :

1. Calcul du barycentre de l'image obtenue après l'application du filtre de Canny :
    - Le barycentre se situe sur la grille, car sa taille et son nombre de pixels influencent fortement l'image.
    - Le barycentre est calculé de manière similaire à son utilisation en physique.
2. Itération en spirale autour du barycentre, en ignorant les points isolés jusqu'à atteindre un pixel blanc :
    - Ce pixel sera nécessairement une lettre, puisque le barycentre est à l'intérieur de l'image et que les lignes de la grille ont été effacées par le filtre de Canny.
3. Récupération de la boîte englobante minimale (bounding box) pour cette lettre à partir du pixel obtenu et de l'image avant l'application du filtre de Canny.
4. Ajustement de cette bounding box afin qu'elle soit carrée (en prenant la taille du plus grand côté) tout en restant centrée.
    - On l'agrandit légèrement pour s'assurer qu'elle contient bien la lettre dans l'image après application du filtre de Canny.
5. Déplacement de cette bounding box depuis la première lettre trouvée dans les quatre directions (haut, bas, gauche et droite) sur l'image après application du filtre de Canny. En se décalant progressivement d'une taille de lettre, nous continuons jusqu'à ce que la densité de la bounding box cesse d'augmenter ou dépasse un certain seuil. Si aucune lettre n'est trouvée dans une direction après 2,5 fois la taille de la lettre, cela indique que nous avons atteint un bord de la grille.
6. Détermination des quatre extrémités :
    - Une fois ces coordonnées obtenues, elles permettent de définir la bounding box de la grille.
7. Calcul des positions des lettres dans la grille :
    - Grâce à la bounding box ainsi qu'au nombre total de lettres dans la grille obtenues lors de la recherche des extrémités $((nb\_haut + nb\_bas + 1) * (nb\_gauche + nb\_droite + 1))$, nous déterminons les positions des lettres dans la grille.

Bien que cette méthode semble complexe en raison du grand nombre d'étapes impliquées, elle n'est pas particulièrement gourmande en ressources informatiques.

5.0.10 Détermination des coordonnées des différents éléments

Après la détection des lignes, nous déterminons les coordonnées de la grille et de la liste de mots en identifiant les intersections des lignes détectées. Cela permet de localiser les cellules de la grille et d'extraire les lettres, ainsi que de localiser la liste de mots.

5.0.11 II. Découpage

Le mode verbose (-v) est particulièrement utile pour vérifier la détection correcte de toutes les parties.

Le découpage de l'image suit les étapes suivantes :

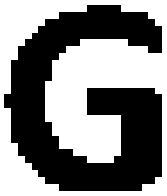
1. Récupération des listes de coordonnées de la grille, de la liste de mots, des mots individuels, et des lettres.
2. Récupération de la zone correspondante à l'aide de la bibliothèque SDL.
3. Sauvegarde au format .bmp dans un dossier "result".

Les fichiers sauvegardés suivent une convention de nommage claire :

- grid.bmp : image de la grille

- letterGrid_1.bmp, letterGrid_2.bmp, etc. : lettres de la grille

- words.bmp : liste des mots



- word_1.bmp, word_2.bmp, etc. : mots individuels

- letterWord1_1.bmp, letterWord1_2.bmp, etc. : lettres de chaque mot

Cette organisation facilite la récupération ultérieure par le réseau neuronal.

Bibliographie

[1] Simple DirectMedia Layer (SDL2). *Official SDL Website*. https://www.libsdl.org/

[2] Canny, J. (1986). A Computational Approach to Edge Detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6), 679-698.

[3] Duda, R. O., Hart, P. E. (1972). Use of the Hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1), 11-15.

[4] Shapiro, L. G. Stockman, G. C. (2001). Computer Vision. Prentice Hall.

[5] Sobel, I., Feldman, G. (1968). A 3x3 Isotropic Gradient Operator for Image Processing. *Stanford Artificial Intelligence Project (SAIL)*.

[6] Bradley, D., Roth, G. (2007). Adaptive Thresholding using the Integral Image. *Journal of Graphics Tools*, 12(2), 13-21.

[7] Mori, S., Nishida, H., Yamada, H. (1999). Optical Character Recognition. John Wiley Sons.

[8] Gonzalez, R. C., Woods, R. E. (2018). Digital Image Processing (4th ed.). Pearson.