# Rapport de soutenance 1:
# A votre C-rvice

Pierre Mittelbronn, Evan Rieber,

Quentin Ruhf, Quentin Schneider

# Rôle de chaque membre de l’équipe

Pierre Mittelbronn :

Ce projet est une expérience totalement nouvelle pour moi. En effet, c’est la première fois que je participe à un travail d’équipe en C impliquant une phase de réflexion aussi importante avant de commencer à coder. C’est un exercice très formateur, car il ne s’agit pas seulement de coder, mais de comprendre la logique et les contraintes du projet. De plus, le fait qu’il y ait peu de documentation rend ce projet difficile, mais intéressant, en effet, nous devons chercher par nous-mêmes et réfléchir à l’algorithme derrière, ce qui est très intéressant.

Dans cette équipe de quatre personnes, mon rôle principal est de concevoir l’interface utilisateur ainsi que le solver. C’est la première fois que je manipule une interface graphique en C, ce qui représente un véritable défi. Je découvre de nouveaux outils

Evan Rieber :

Dans l’équipe, je suis principalement chargé de la conception et de la programmation du réseau de neurones. Bien que je ne sois pas habitué à ce style de développement, en effet, je n'ai jamais développé de réseau de neurones auparavant, donc ce projet est un excellent moyen d'apprendre en détail le fonctionnement des réseaux de neurones.

Ce projet me tient particulièrement à cœur, car les réseaux de neurones et l'IA en général sont des sujets qui m'intéressent particulièrement, car ce sont des sujets complexes, mais qui permettent de faire beaucoup de choses différentes en fonction des besoins.

Quentin Ruhf :

Ce projet, nouveau en son genre pour moi, est particulièrement intéressant. En effet, je n'avais encore jamais réalisé de projet en lien avec les réseaux de neurones ou la vision par ordinateur. Ce nouveau type de projet est très motivant et me permet de développer de nombreuses connaissances.

Il se distingue également par le peu de ressources disponibles, la résolution de mots cachés à l'aide de l'OCR est un domaine très spécifique, pour lequel il existe peu de références. Cela rend le défi encore plus stimulant, car il m'a conduit à concevoir de nouveaux algorithmes adaptés à cette problématique.

Dans ce projet, ma principale tâche consiste au prétraitement des données, ce qui m'a amené à concevoir plusieurs algorithmes pour la détection de grille, le nettoyage du bruit et bien d'autres aspects techniques.

Ce projet m'offre donc une excellente occasion de développer à la fois mes compétences techniques et ma capacité d'analyse approfondie. De plus, il comporte une importante partie de gestion et de travail en équipe, notamment à travers la répartition des tâches et le maintien d'une structure de code propre, claire et fonctionnelle entre tous les membres du groupe.

Quentin Schneider :

Ce projet m'a déjà beaucoup apporté grâce à sa nouveauté et une équipe très impliquée dans ce qu'elle fait et avec qui je prends plaisir à travailler.

En effet, c'est la première fois que je travaille avec le traitement d'image ainsi qu'avec un réseau de neurones et vu l'importance des réseaux de neurones de nos jours, je trouve cela très intéressant d'avoir un premier aperçu sur la conception de celui-ci.

Le cahier des charges donné ainsi qu'un projet concret permet de s'immiscer d'une certaine manière dans le monde du travail, ce que j'apprécie. Dans ce projet, mon objectif était la création du réseau de neurones ainsi que l'optimisation de différentes parties du code.

# Pilotage du projet

Nous verrons dans cette partie la progression et la finalisation de notre projet OCR, en passant par nos objectifs, la planification des tâches ainsi que l'avancement de notre projet pour cette soutenance finale par rapport à ce qui était attendu.

## Objectifs

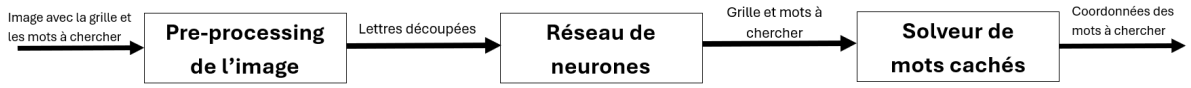
Pour cette soutenance finale, notre objectif était de continuer à consolider les bases solides que nous avions construites lors de la première période de notre travail. En effet, lors de cette première phase, nous avions mis en place l'ensemble du système de communication permettant l'échange entre les fichiers image, les bibliothèques comme SDL, ainsi que les structures personnalisées nécessaires au bon fonctionnement de nos algorithmes.

Nous avions déjà bien avancé sur le pré-processing des images afin d'obtenir des entrées exploitables pour notre réseau de neurones. Lors de cette deuxième partie, nous avons poursuivi le développement en finalisant les dernières fonctionnalités : notamment la détection complète des mots dans la grille et la mise en place finale de notre réseau de neurones capable de reconnaître les lettres de l'alphabet à partir des images prétraitées.

Enfin, une étape essentielle de cette phase a été l'intégration de l'ensemble de nos modules autour d'une interface unique, réalisée grâce à GTK. Cette interface permet désormais de centraliser tout le processus : de l'importation de la photo de la grille de mots, au pré-processing, à la reconnaissance des lettres par le réseau de neurones, et enfin à la recherche automatique des mots dans la grille. L'objectif principal a été atteint : transformer notre prototype en un système fonctionnel et cohérent, où chaque étape communique efficacement avec les autres, tout en offrant une interface utilisateur claire et intuitive.

## Planification des tâches

Comme dit précédemment, le projet a été découpé en trois tâches principales. C'est trois tâches ont pu être développées en parallèle étant donné que chaque tâche sont indépendantes les unes des autres. Comme le montre le schéma ci-dessous, les trois blocs principaux peuvent facilement être liés les uns aux autres, car nous savons quels sont les paramètres et les valeurs de retour de chacun de nos blocs.



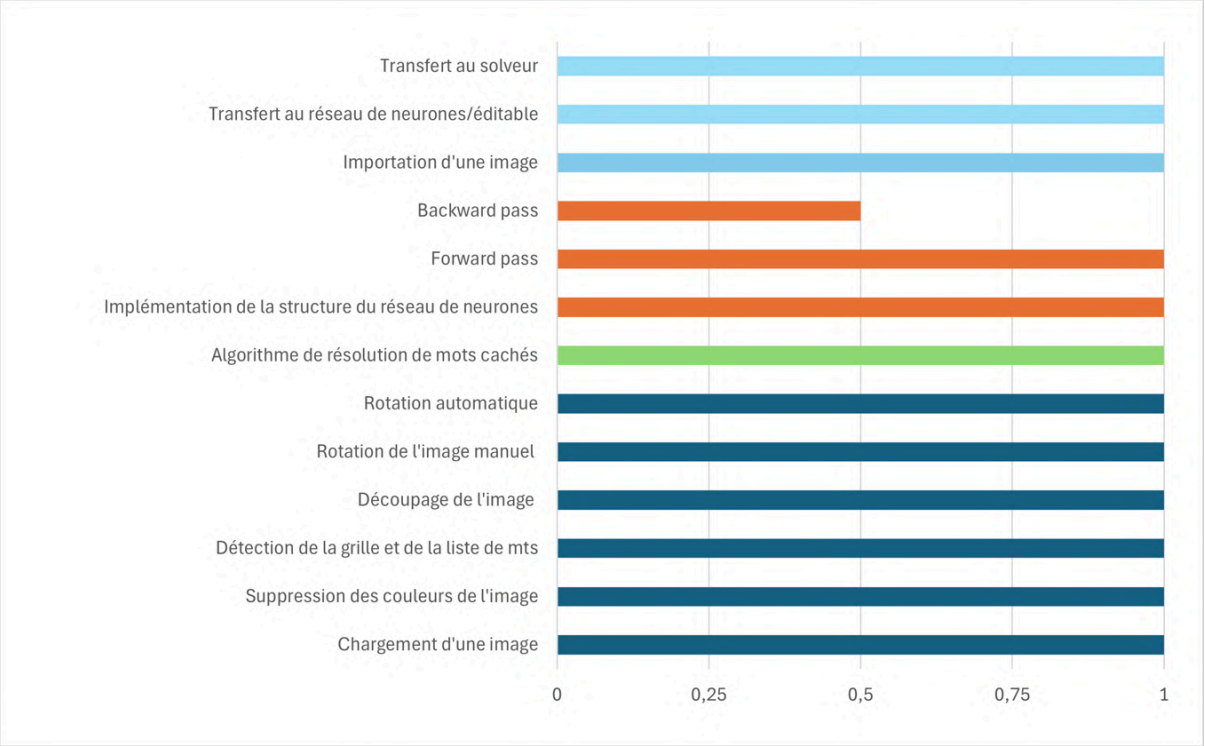

Pour cette première soutenance, les fonctions faisant les liens ne sont pas encore implémentés, comme celle qui va passer de la lettre découpée sur l'image à sa correspondance pour que cette dernière soit passée en argument à notre solveur, après avoir été regroupée avec les lettres adjacentes. Cependant, tous les blocs sont déjà présents pour cette première soutenance comme demandé. Pour la soutenance finale, voici les tâches qu'il nous reste à faire et à répartir en fonction de la difficulté de ces dernières :

- Création de l'interface utilisateur
- Lier les différents blocs comme expliqué précédemment
- Implémenter la rotation automatique
- Adapter le réseau de neurones à la reconnaissance de lettres
- Résolution de la grille
- Affichage des réponses

## État d'avancement du projet

Voici un graphique permettant d'avoir une vue globale de notre avancement pour cette première soutenance par rapport à nos objectifs fixés.

La première partie en bleu correspond à la partie du pre-processing. La deuxième en vert correspond à la partie du solveur de mots cachés. La dernière partie en orange correspond au réseau de neurones.




## Pre-processing de l’image : (avancement 10/10)

L’ensemble du pré-traitement a été finalisé, le programme est capable de tourner l’image, supprimer le bruit, mettre en noir et blanc, détecter à la fois la grille et la liste de mots. En plus de tout cela, la partie pré-traitement peut envoyer des informations au réseau de neurones et est donc complètement finalisée.

## Solveur de mots cachés : (avancement 10/10)

Le solveur de mots cachés est implémenté et optimisé.

Il ne connaîtra que très peu voir aucun changement jusqu'à la soutenance finale.

## Le réseau de neurones : (avancement 7/10)

Les problèmes de back propagation de la première soutenance ont persisté pour cette seconde soutenance malgré que les autres parties comme le feed forward ou la structure de notre réseau sont bien implémentés.

### La structure interne : (avancement 10/10)

Nous avons choisi d'implémenter une structure de neurones avec 2 couches cachées et respectivement 70 et 50 neurones. Nous avons pris cette décision car après plusieurs recherches, nous en avons conclu que c'était la disposition optimale pour la recherche de caractères. Notre set d'entraînement est composé des 26 lettres de l'alphabet en 750 exemplaires différents car nous ne voulions pas prendre les lettres extraites pour set d'entraînement de notre réseau de neurones. Cependant, nous avons peut-être vu trop grand pour cette partie ce qui nous a désavantagé pour la suite.

### Le "forward path" : (avancement 10/10)

Le "forward path" correspond au chemin avant dans un réseau de neurones, partant dans notre cas d'une image de lettre et arrivant sur une prédiction du caractère correspondant. Sur ce point, c'est le nombre d'opérations qui est important, car le nombre d'opérations réalisées est exponentiellement proportionnel au nombre de neurones du réseau. Nous avons dû modifier cette partie pour passer par une output avant passage dans notre fonction d'activation car sans ça, nous ne pouvions pas avoir une backpropagation propre. Cette partie fonctionne en donnant les bonnes.

### Le "backward pass" : (avancement 5/10)

Le "backward path" correspond à la "backpropagation" donc à l'algorithme permettant de rendre le réseau de neurones plus juste et précis. Il n'est pour l'instant pas viable pour un réseau de neurones plus gros que celui permettant de faire la porte logique présentée plus tôt. Ce choix a été fait pour des raisons qui seront expliquées dans la partie prévue à cet effet.

## L'interface graphique : (avancement 10/10)

L'interface graphique est implémentée et permet d'importer une image, de modifier la rotation si la rotation auto ne marche pas correctement, puis de modifier les sorties du réseau de neurone en cas d'erreur et renvoie l'image résolue.

# Difficultés et choix

## Solveur

Le solveur a pour objectif de récupérer la position d’un mot sur la grille de mots mêlés. Son rôle est crucial, sans ça il est impossible de résoudre la grille.

### Gestion de la matrice

Dans un premier temps, il faut commencer par convertir le fichier, contenant la matrice représentant la grille de mots mêlés, donné en paramètre, en un tableau de double pointeur.

Pour cela, on commence par lire le fichier avec fopen de la librairie stdio.h, on crée un tableau dynamique avec un buffer de 12 sur la première ligne, puis on parcourt tous les caractères avec fgetc jusqu’à tomber sur EOF, dès qu’on tombe sur un caractère, on le met en majuscule, si le nombre de caractères de la ligne est supérieur au buffer alors on realloc en incrémentant la capacité de la ligne d'un jusqu’à tomber sur \n, à ce moment-là, on définit le buffer à la valeur du compteur du nombre d’éléments dans la ligne, étant donné que la grille de mot croisé est une matrice et contient donc en théorie toujours le même nombre de caractères sur une ligne cela devrait nous donner la taille optimale pour chaque ligne.

Une fois avoir tout parcouru, on n'oublie pas de free tout hors la matrice et de fclose.

## Algorithme de résolution

On commence par récupérer la matrice précédemment créée, on commence par récupérer le nombre et la taille des lignes, puis on parcourt notre matrice jusqu'à trouver une lettre correspondant à la première lettre du mot à trouver, en vérifiant à chaque fois les bornes pour éviter tout dépassement. Dès qu'on trouve une lettre correspondante, on regarde tout autour de la lettre si une des lettres correspond à la deuxième lettre à trouver, si c'est le cas, on regarde récursivement jusqu'à arriver au bout du mot, si on a trouvé le mot, on renvoie un tableau de deux éléments, correspondant au coordonné x et y de fin, sinon on renvoie -1. Afin de ne faire qu'une seule fonction récursive, on n'incrémente pas à l'aide des opérateurs ++ et -- mais avec x += inc_x, donc si on veut décrémenter, on dit que inc_x = -1 et si on veut incrémenter, on dit inc_x = 1 et si on ne veut pas y toucher, on dit juste inc_x = 0.

Une fois le mot trouvé, on ne parcourt pas toute la matrice inutilement, on return sans oublier de tout free avant pour éviter les memories leaks.

# Pre-Processing

Dans ce projet, le travail de prétraitement joue un rôle essentiel. En effet, c'est ce prétraitement qui détermine en grande partie la qualité du résultat final.

L'objectif est à la fois multiple et complexe, il s'agit de traiter chaque image, d'appliquer si nécessaire une rotation, de la convertir en niveaux de gris puis en noir et blanc, mais aussi de nettoyer les bruits visuels. À cela s'ajoute la détection et l'extraction des données pertinentes, qui seront ensuite transmises au réseau de neurones pour la phase d'analyse.

## Gestion des Images

Dans un premier temps, afin de simplifier la gestion des images, nous avons mis en place une abstraction **Image**, représentée par une structure (*struct*) composée d'une taille et d'une liste de pixels.

Chaque pixel est lui-même une structure contenant uniquement les informations utiles au prétraitement, telles que la couleur, les coordonnées ou d'autres données pertinentes.

Nous disposons d'un ensemble de fonctions permettant d'exploiter efficacement cette abstraction.

Pour charger les images, nous utilisons la bibliothèque SDL, qui permet d'importer une image sous forme de **SDL_Surface**.

Deux fonctions assurent ensuite la conversion entre **SDL_Surface** et notre structure **Image**, et inversement.

Ainsi, SDL est utilisée exclusivement pour le chargement initial, tandis que toute la suite du prétraitement s'effectue via notre abstraction **Image**.

Nous avons également développé plusieurs fonctions supplémentaires, notamment pour exporter une image, récupérer une sous image, changer la couleur des pixels, etc.

Cette abstraction rend la manipulation des images plus simple, plus claire et plus performante en réduisant la consommation mémoire.

## Suppression des couleurs

### Niveau de gris

La première étape essentielle du traitement d'image est la binarisation, c'est-à-dire la conversion de l'image en noir et blanc.

Avant cela, il est nécessaire de transformer l'image en niveaux de gris, ce qui se fait à partir de la luminance selon la formule suivante :

$$Gris = (Rouge \times 0.2125) + (Vert \times 0.7154) + (Bleu \times 0.0721)$$

Cette formule, très simple à mettre en œuvre, permet de convertir chaque pixel en une valeur de gris. C'est la première étape du prétraitement.

### Binarisation

**Première approche : seuil global**

Pour la binarisation, une première méthode consiste à utiliser un seuil fixe.

Celui-ci peut être calculé, par exemple, en prenant la moyenne des niveaux de gris de tous les pixels.

Chaque pixel est alors comparé à ce seuil :

- si sa valeur de gris est supérieure au seuil, il devient noir
- sinon, il devient blanc

Cette approche, bien que simple à implémenter, ne fonctionne que dans des cas idéaux, lorsque l'image contient peu de bruit ou d'ombres. Dès que des variations lumineuses apparaissent, cette méthode perd une grande quantité d'informations utiles ou conserve, au contraire, des pixels indésirables.

**Méthode avancée : l'algorithme de Sauvola**

Pour pallier ces limites, nous avons choisi d'utiliser l'algorithme de **Sauvola**, qui repose sur une analyse locale plutôt que globale.

Cet algorithme calcule un seuil spécifique à chaque pixel, en se basant sur la moyenne et l'écart-type des pixels situés dans un voisinage de taille fixe **k**.

La formule de Sauvola est la suivante :

$$seuil = moyenne_{voisin} \times (1 + c \times (\frac{écart\_type_{voisin}}{R} - 1))$$

où :

- $moyenne_{voisin}$ = moyenne du niveau de gris k pixels voisin
- $ecartType_{voisin}$ = écart-type des k pixels voisin
- $c$ = constante comprise entre 0.2 et 0.5 (par défaut 0.5)
- $R$ = plage dynamique de l'écart-type (par défaut 128)

Grâce à cette approche locale, l'algorithme de Sauvola permet de binariser les images de manière plus robuste, en tenant compte des bruits, des ombres et des variations d'éclairage, pour obtenir un résultat beaucoup plus fiable et homogène.

**Exemples de résultat :**

IMAGINE
RELAX
COOL
RESTING
BREATHE
EASY
TENSION
STRESS
CALM

| P | X | U | T | S | I | N | I | U | P | R | V | G | B | M | D | D |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| E | H | A | A | S | P | O | J | P | E | T | B | E | Q | Z | L | C |
| A | U | N | T | E | G | Q | T | L | H | R | Z | F | A | T | O | P |
| S | H | X | F | N | G | U | A | X | E | A | A | Y | P | O | M | H |
| Y | O | Y | Y | L | D | X | L | A | K | Y | U | Z | L | B | S | K |
| J | X | M | U | U | G | Q | T | R | I | M | A | G | I | N | E | B |
| H | F | N | W | F | X | H | D | P | B | B | B | T | N | V | S | K |
| H | I | I | H | D | E | S | Q | F | U | M | Y | E | R | N | S | X |
| R | P | B | Z | N | H | S | D | S | L | H | O | N | B | S | S | S |
| E | H | X | A | I | Z | I | H | A | H | O | E | S | Q | F | E | F |
| C | W | Z | I | M | V | D | C | J | V | S | S | I | M | G | R | W |
| L | A | I | I | R | Z | Q | Q | H | X | D | Z | O | Z | Q | T | R |
| W | C | A | X | E | Z | R | G | H | A | I | Z | N | E | C | S | E |
| B | R | H | F | O | T | G | N | I | T | S | E | R | E | O | V | Z |
| M | W | V | W | Q | D | U | I | H | W | Q | T | S | B | I | M | L |
| T | D | T | O | N | Z | C | X | X | R | G | E | L | K | H | F | Q |
| Q | N | E | K | S | V | M | O | T | F | A | L | A | A | E | W | B |

| A | A | S | S | E | M | B | L | Y | O | O | Y | H | O |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| H | D | I | B | O | V | P | D | S | H | T | Y | H | B |
| J | L | L | M | M | T | D | Y | L | E | O | P | R | L |
| L | B | O | R | S | A | C | O | P | L | Y | C | U | A |
| V | A | A | M | O | J | O | I | P | Y | O | Y | S | Y |
| T | S | M | A | I | W | S | M | S | T | L | R | T | A |
| P | L | M | L | H | Y | S | J | S | A | L | O | A | V |
| P | H | P | H | I | P | T | R | U | W | B | T | H | A |
| R | W | M | R | H | T | M | L | T | J | P | Y | S | J |
| H | A | T | L | T | I | D | E | S | L | H | O | M | A |
| O | U | S | L | L | B | J | A | O | J | O | Y | O | B |
| O | O | S | O | S | H | M | L | M | H | S | L | T | T |
| S | R | T | A | P | Y | T | H | O | N | Y | A | A | L |
| P | N | N | H | E | L | L | O | H | M | O | O | L | V |

HELLO
WORLD
RUST
PHP
JAVA
ASSEMBLY
PYTHON
HTML
MOJO
BASIC

## Nettoyage de l’image

Pour nettoyer l’image, nous nous appuyons principalement sur un algorithme de détection de formes basé sur la coalescence de pixels.

L’objectif est de parcourir l’image afin d’en extraire l’ensemble des formes correspondant à des groupes de pixels connectés.

Ainsi, chaque lettre sera identifiée comme une forme distincte, tout comme les bruits visuels, qui pourront ensuite être filtrés.

**Représentation des formes :**

Pour cela, nous avons défini une structure **Shape** contenant :

- un ensemble de pixels appartenant à la forme
- le nombre total de pixels de la forme
- ses bornes minimales et maximales en $x$ et $y$, c’est-à-dire le rectangle englobant (bounding box)

Un ensemble de fonctions permet ensuite de calculer diverses propriétés d’une forme, telles que :

- sa hauteur et sa largeur
- son aire
- sa densité
- et son rapport d’aspect (aspect ratio)

La fonction principale, chargée d’extraire toutes les formes d’une image, repose sur un parcours en profondeur (DFS).

Chaque pixel contient un pointeur vers la forme à laquelle il appartient, garantissant qu'un pixel n'est associé qu'à une seule forme.

**Fonctions de manipulation :**

Nous avons également implémenté plusieurs fonctions utilitaires, permettant, par exemple, de supprimer une forme d'une image ou de modifier la couleur de tous les pixels appartenant à une forme donnée.

Ces outils facilitent le traitement et la visualisation des différentes étapes du nettoyage d'image.

**Étape de filtrage**

Une fois les formes extraites, nous procédons à un filtrage progressif :

1. Suppression des petites formes :

   Les formes contenant très peu de pixels sont éliminées.

   En effet, les pixels isolés ou les petits groupes de pixels sont généralement dus au bruit de l'image. Ce premier filtre simple permet donc de les supprimer efficacement.

2. Suppression des formes trop grandes :

Nous utilisons ici l'aire des formes, en définissant un seuil basé sur un certain percentile.

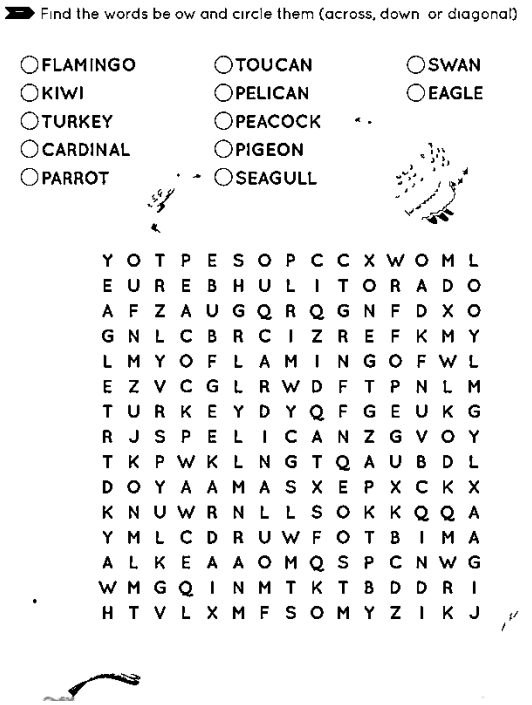
Ce filtre élimine notamment les grilles présentes sur certaines images.

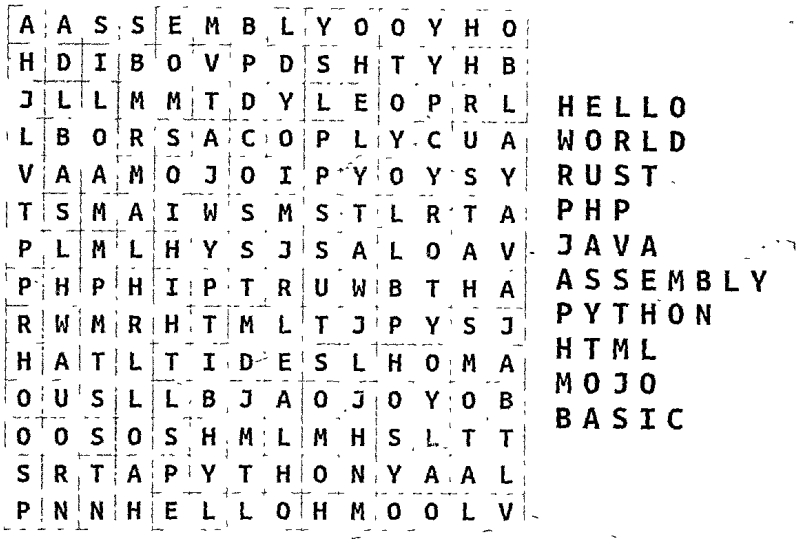
Ce choix garantit une homogénéité de traitement, car l'algorithme ne dépend pas de la présence ou non d'une grille dans l'image d'origine.

3. Filtrage par rapport d'aspect :

   Enfin, nous supprimons les formes dont le rapport d'aspect est trop élevé ou trop faible, ce qui permet d'éliminer, par exemple, les lignes trop allongées ou les formes incohérentes.

Voici des exemples des images après filtrage :




## Détection de la position de la grille

**Préparation de l'image**

Nous utilisons la liste des formes obtenue après l'étape de nettoyage décrite précédemment.

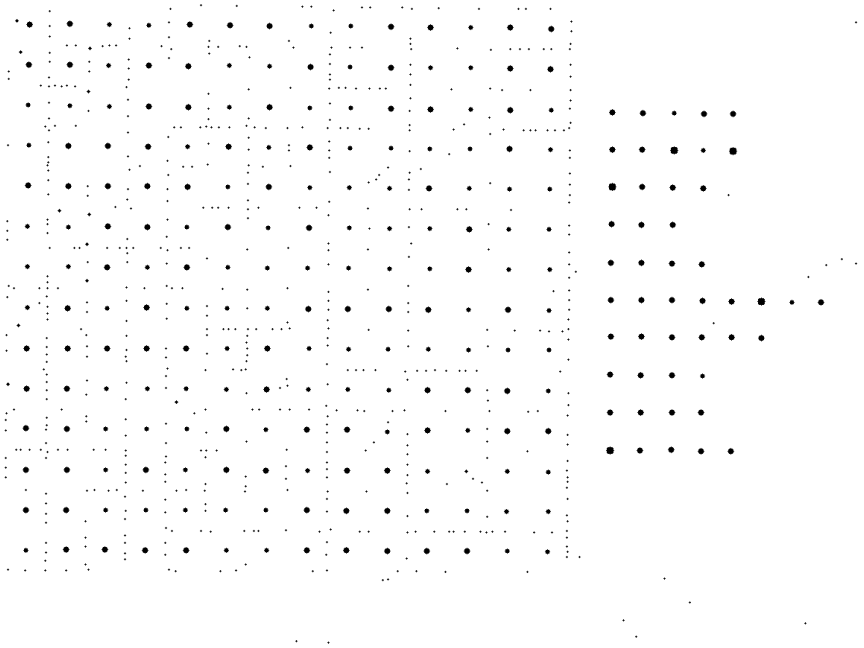
À partir de ces formes, nous générons une nouvelle image dans laquelle chaque forme est remplacée par un cercle de pixels noirs, dont la taille est proportionnelle à la densité de la forme d'origine.

Chaque cercle est placé au centre de la forme correspondante.

De cette manière, nous obtenons une image simplifiée où :

- les lettres sont représentées par des cercles de taille moyenne à grande, répartis de manière régulière
- les bruits résiduels ou petites formes parasites deviennent de petits points dispersés

Exemple de génération de l'image :



**Détection des droites avec la transformée de Hough**

Une fois cette image générée, nous appliquons la transformée de Hough pour détecter les droites présentes dans l'image.

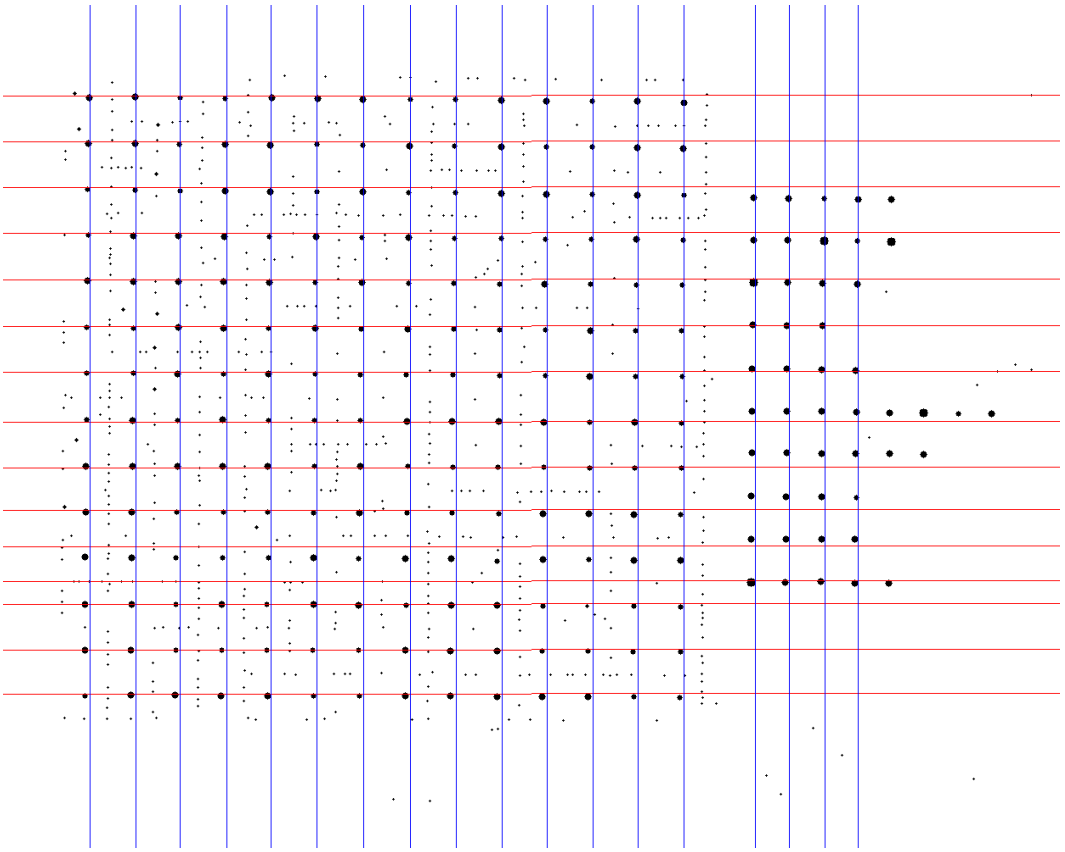
Étant donné le faible nombre de pixels, les droites détectées regroupent principalement les points correspondant aux lettres, tandis que les bruits, souvent non alignés, ne génèrent pas de droites significatives.

Nous appliquons ensuite un premier filtre simple pour ne conserver que les droites horizontales et verticales, en autorisant une petite marge d'erreur angulaire.

Les droites sont ensuite regroupées en deux ensembles :

- les droites majoritairement horizontales
- les droites majoritairement verticales

Résultat de la détection initiale :



**Rotation automatique**

Sur le même procédé de la transformée de Hough, au départ, après avoir chargé l'image et l'avoir transformée en noir et blanc, on va essayer de trouver les droites présentes sur l'image. Ensuite, à l'aide de ces droites, on va garder les plus représentatives, ce qui va permettre de récupérer l'angle moyen des droites. Une fois cet angle récupéré, on va pouvoir appliquer une rotation. Si, malgré cette rotation automatique, l'image ne convient toujours pas à l'utilisateur, il a la possibilité d'appliquer une rotation manuelle pour affiner les résultats.

**Filtrage et raffinement**

Comme on peut le voir, les droites détectées encadrent globalement bien la grille, mais certaines droites parasites subsistent notamment dans la zone de la liste de mots.

Pour affiner le résultat, nous appliquons un second filtre basé sur l’espacement moyen entre les droites.

Lorsqu’un espacement anormalement grand est détecté, les droites en excès sont supprimées.

Ce traitement permet de ne conserver que les lignes appartenant réellement à la grille.

**Extraction finale de la grille**

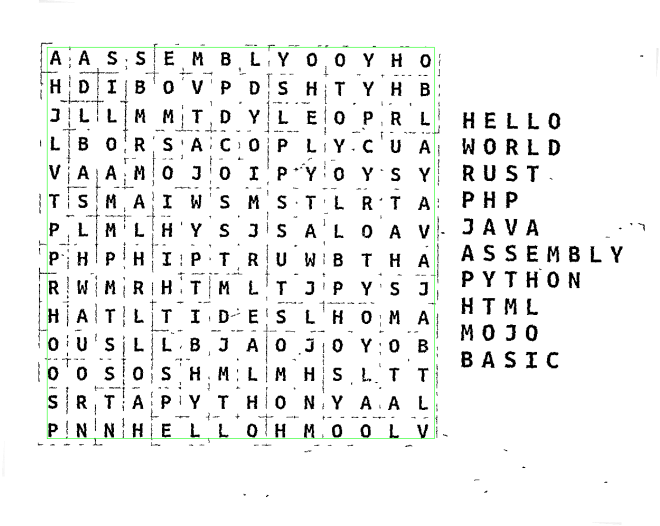
Une fois les droites correctement identifiées, nous déterminons les points d'intersection les plus extrêmes pour déduire les coordonnées de la grille.

Ces coordonnées sont ensuite étendues légèrement, en fonction de la moyenne de la largeur et de la hauteur des formes détectées, afin de compenser le fait que les cercles sont centrés sur les lettres.

Cette étape permet d'englober l'intégralité de la grille de lettres.

Nous pouvons alors extraire la sous-image correspondant à la grille, qui servira pour les étapes suivantes de détection des cellules et reconnaissance de caractères.

Exemple final de détection de la grille :




H C A M P A I G N S L Z
Y C P H Q M L C A B E E
U W O R K D R I V E K T
K O Y O D Q U G Y E A F
S A L E S I Q P R W S Q
E T U M B L J P E Z X C
D M A I L C A Y N U O R
P R K Y U G P E O P L E
H U H G B O A V J L M A
P K Y H A D C R M R S T
A L M U L A A U X S T O
K R U B O O K S A T O R

CAMPAIGNS
ULAA
CRM
ONE
DESK
SURVEY
CREATOR
BOOKS
PEOPLE
MAIL
SALESIQ
WORKDRIVE
CLIQ

S U M M E R L H
C I P O R T N O
B S U N B A L L
R E L A X E P I
T D A Q S A N D
A Y B C A Z I A
N F U N H R S Y

TROPIC
BEACH
SUMMER
HOLIDAY
SAND
BALL
TAN
RELAX
SUN
FUN

## Détection et extraction des image de la grille

La dernière étape consiste à extraire une image pour chaque cellule de la grille.

**Identification des formes pertinentes**

Une fois la sous-image de la grille récupérée, nous réutilisons la fonction décrite précédemment pour détecter l'ensemble des formes présentes dans cette image.

À ce stade, un filtrage renforcé est appliqué afin de supprimer presque entièrement les bruits résiduels.

Pour chaque forme détectée, nous analysons la distribution des formes voisines :

- horizontalement (sur la même ligne)
- et verticalement (sur la même colonne)

Cette analyse repose sur la bounding box (rectangle englobant) de chaque forme.

En effet, les lettres sont alignées à la fois horizontalement et verticalement dans la grille.

Ainsi, une forme qui présente peu de voisins sur sa ligne ou sa colonne peut être considérée comme un bruit isolé et est donc supprimée.

**Double passe de filtrage**

L'algorithme effectue deux passes successives :

Première passe :

Un premier seuil minimal de voisinage est appliqué pour identifier les formes susceptibles d'appartenir à une lettre.

Deuxième passe :

Après cette première sélection, nous re-calculons le nombre moyen de formes par ligne et par colonne.

Un second filtrage est ensuite appliqué, cette fois avec un seuil plus strict, adapté aux caractéristiques de la grille détectée.

À la fin de ce processus, la quasi-totalité des bruits visuels est éliminée, et il ne subsiste que les formes correspondant réellement aux lettres de la grille.

**Gestion des collisions et extraction finale**

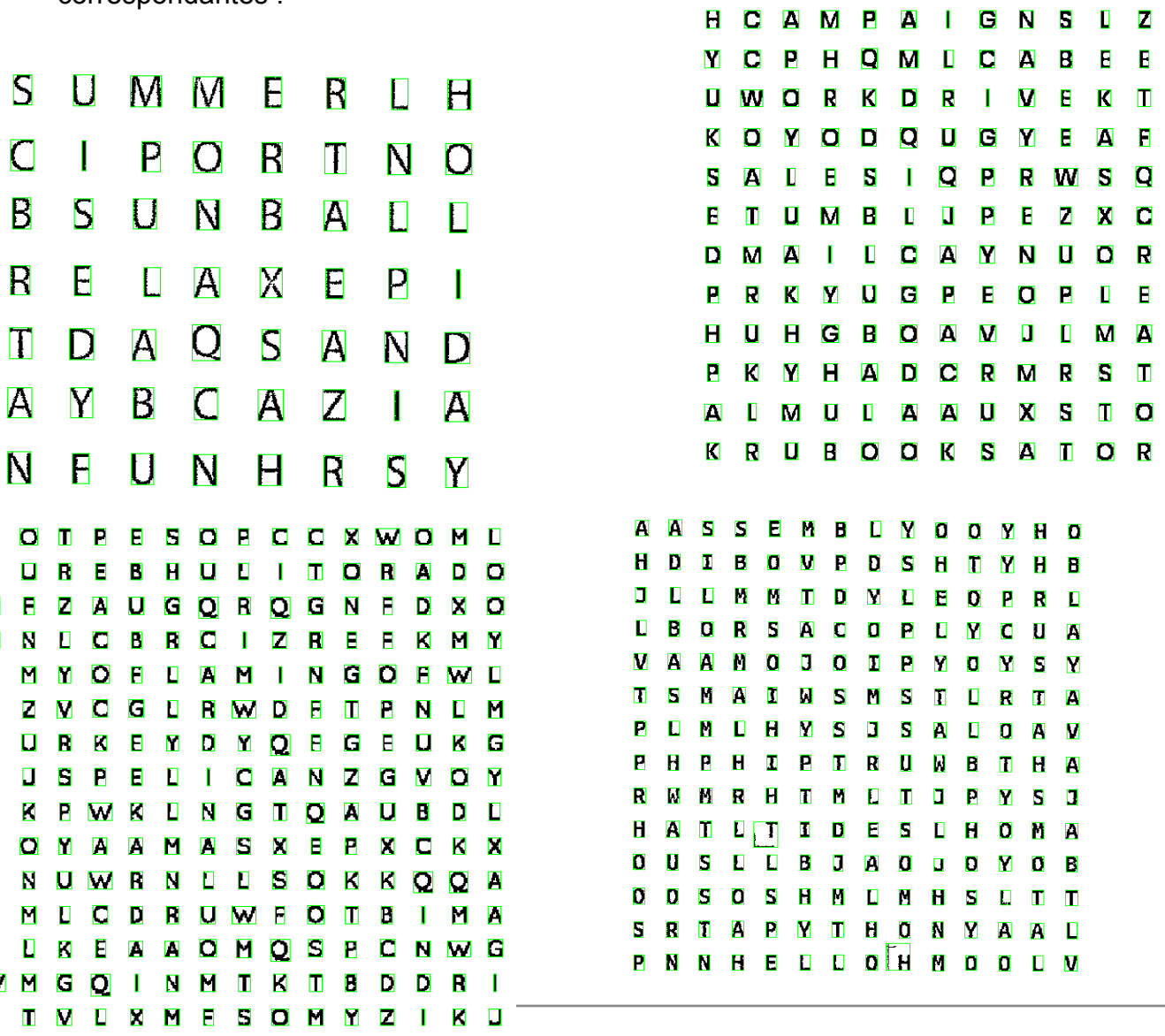
Pour garantir la cohérence du résultat, un mécanisme de vérification est mis en place :

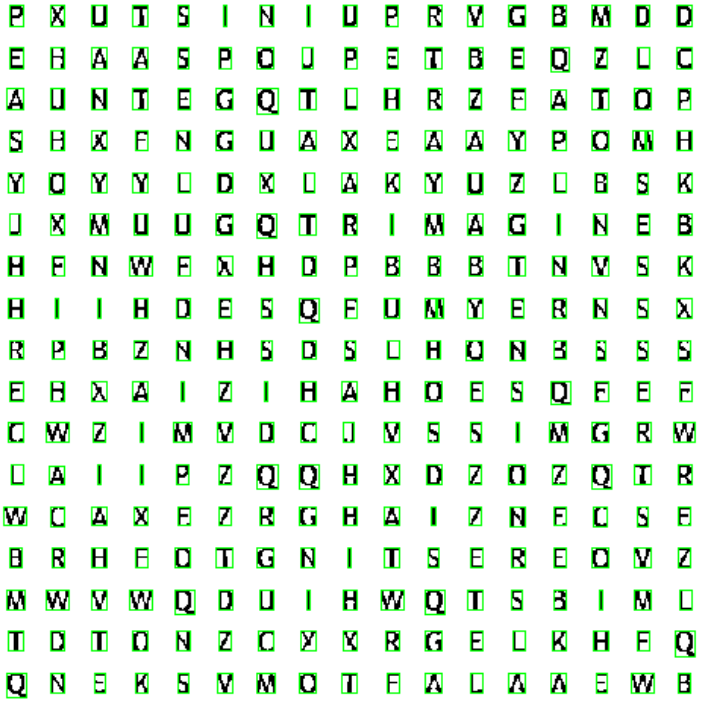
si deux formes sont trop proches l'une de l'autre et que l'algorithme détecte que la même case est déjà occupée, il sélectionne automatiquement la forme la plus proche du centre de la cellule correspondante.

Enfin, les sous-images correspondant à chaque forme sont extraites individuellement à partir de leurs coordonnées précises.

Ces sous-images serviront ensuite à la reconnaissance des caractères par le réseau de neurones.

Voici un exemple illustrant les cellules détectées avant l'extraction des sous-images correspondantes :





Nous profitons de cette étape pour créer un fichier texte afin de connaître les coordonnées de chaque cellule dans la grille. Chaque lettre de la grille, donc la cellule r_c, va être mappée à une coordonnée x, y qui représente ses coordonnées dans l'image initiale. Cela va notamment être utile à la suite du réseau de neurones : une fois le résultat trouvé et les coordonnées des mots déterminées, nous pouvons tracer un vecteur pour afficher le résultat dans l'image originale.

## Détection de la liste de mots

Pour la détection de la liste de mots, nous commençons par supprimer la grille de l'image principale. Ensuite, nous réappliquons les fonctions de filtre vues précédemment.

Comme dans la majorité des algorithmes, nous utilisons les algorithmes de pixels coalescents pour récupérer l'ensemble des formes et donc toutes les lettres présentes sur l'image.

Les formes sont ensuite regroupées en mots en se basant sur leur position verticale (pour déterminer la ligne) et leur espacement horizontal (pour séparer les mots). Les lettres de chaque mot sont triées par position horizontale, puis chaque mot est ajouté à une liste principale. À la fin, nous obtenons une structure contenant tous les mots détectés dans l'ordre correct. Nous récupérons ainsi un triple pointeur correspondant à l'ensemble des mots et, pour chaque mot, une liste de lettres.

Nous supprimons ensuite les artefacts : si les mots contiennent un seul caractère ou trop de caractères, nous supprimons le mot. Cela évite les problèmes liés aux mots mal détectés.

Pour chaque mot, nous exportons les images de chaque lettre afin de les traiter ensuite avec le réseau de neurones.

Cette étape a posé quelques difficultés. En effet, sur certaines images, les mots ne sont pas parfaitement détectés : il peut arriver qu'il manque une lettre, mais cela est rectifiable par la suite dans notre programme grâce à l'interface. De plus, le bruit présent sur l'image pose aussi un problème : nous ne sommes pas arrivés à le supprimer complètement, ou pour certaines images, nous n'arrivons pas à différencier un bruit d'une lettre. Cependant, notre tri, qui permet de supprimer les artefacts, permet d'atténuer cette problématique.

# Le réseau de neurones

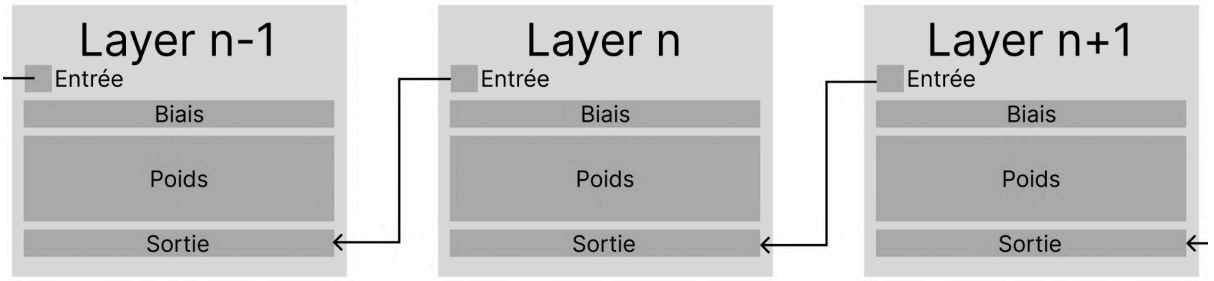
La principale difficulté du réseau de neurones est de construire quelque chose de cohérent et de clair. Pour ce faire, il est important de découper le réseau en 3 parties distinctes, la structure interne donc comment chaque neurone est stocké dans la mémoire, le “forward path” donc le chemin de calcul du résultat du réseau de neurones puis pour finir, le “backward path” le chemin de correction des erreurs du réseau de neurones.

## La structure interne

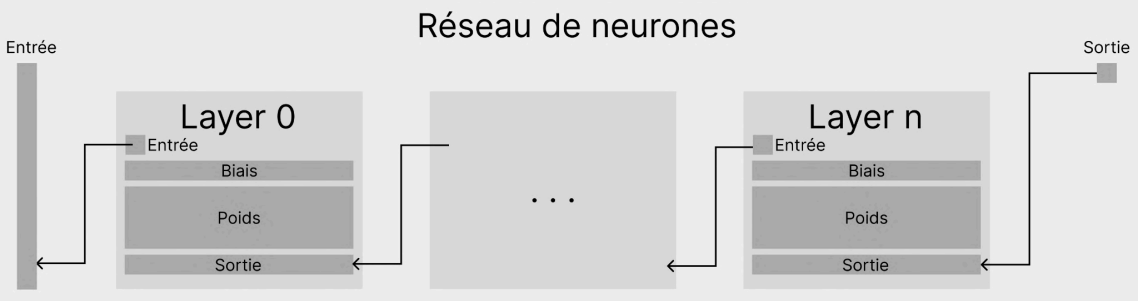
La structure interne du réseau de neurones est assez complexe, car chaque neurone doit pouvoir accéder à la sortie de chaque neurone du niveau précédent, mais également un poids pour chacun d’entre eux et enfin un biais. Dans notre cas, on ne traite pas neurone par neurone, mais par couche de neurones. Pour chaque couche, on a donc besoin des éléments suivants :

- layer_size ⇔ la taille de la couche de neurone
- prev_layer_size ⇔ la taille de la couche de neurone précédente ou égale à la taille de l’entrée du réseau de neurones si la couche est la première du réseau
- biases[layer_size] ⇔ la liste des biais de la taille de layer_size
- weights[layer_size][prev_layer_size] ⇔ liste de liste de taille prev_layer_size de taille layer_size
- Un moyen d’accéder à la liste de lortie de la couche précédente

La structure des couches du réseau de neurones peut donc se schématiser comme ceci :




Autour de ceci nous avons donc besoin d’une structure qui englobe cette liste de couches. Cette structure peut être schématisée tel quel :



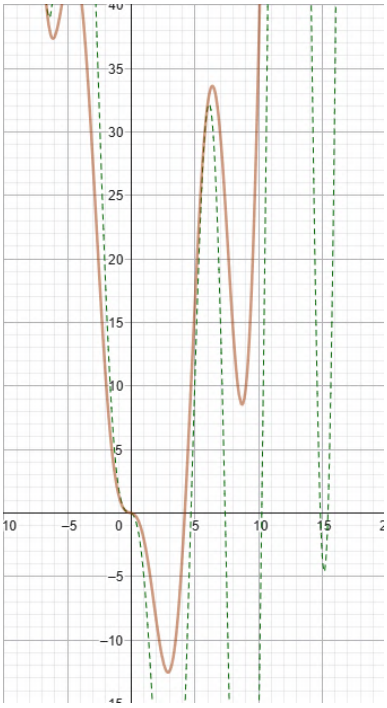

Cette structure permet d'éviter toute redondance de données au sein du réseau de neurones tout en restant la plus lisible possible.

## Le "forward path"

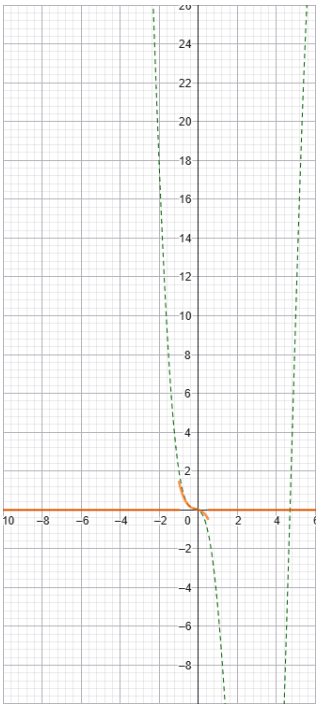
Le "forward path" correspond donc au calcul de la sortie du réseau de neurones, pour ce faire dans l'entrée du réseau, on met la liste correspondant à l'entrée que l'on veut calculer puis pour chaque couche de neurone, il faut, pour chaque neurone de la couche multiplier chaque sortie de la couche précédente avec les poids du neurone puis lui ajouter le biais. La sortie d'un neurone n peut donc de calculer de cette manière :

$$sortie_n = \sum_{k=1}^{prev\_layer\_size} (poid_{n,k} \times entrée_k) + biais_k$$

Après avoir fait ces étapes pour chaque neurone de la couche, il faut encore passer ses sorties brutes dans des fonctions d'activations, qui sont différentes en fonction du rôle de la couche de neurone, en effet si la couche de neurone est une couche interne la fonction d'activation est la "ReLU activation function" (ReLU signifiant Rectified Linear Unit) définie ainsi : $ReLU(sortie) = max(0, sortie)$, cette fonction renvoie donc à zéro si la sortie brute est négative sinon elle la laisse telle quel. Cette fonction permet de faire en sorte qu'un neurone ait un effet sur la sortie uniquement dans certains cas, pour schématiser, un réseau de neurones peut être utilisé pour approximer une fonction.



Dans le cas schématisé par l’image de gauche, on peut voir la courbe en verte que notre réseau de neurones essaie d’approcher, prenons l’exemple d’un neurone qui aurait pour effet d’ajouter la valeur de la fonction marron au résultat du réseau de neurones. Il paraît évident que dans la majorité des cas ce neurone ne donne pas une bonne approximation de la courbe verte sauf lorsqu'on se rapproche de l’origine, le rôle de la fonction ReLU va donc dans ce cas, faire en sorte que neurone ajoute sa solution au problème proche de l’origine, mais nulle part ailleurs, résultat schématisé sur l’image de droite.



Dans le cas où la couche serait la dernière couche du réseau de neurones, la fonction ReLU n'est pas viable. Le réseau de neurones doit renvoyer un pourcentage correspondant à la probabilité que l'entrée soit d'une classe donnée (dans notre exemple, la probabilité pour chaque lettre que l'image soit cette lettre). Dans ce cas, il faut donc une fonction d'activation qui ramène toutes les sorties entre 0 et 1 tout en gardant l'écart entre deux sorties. Ce rôle est donc rempli par la "Softmax activation function" qui est une fonction qui ne prend pas en entrée une sortie de neurone, mais la sortie de tous les neurones de la couche et qui peut se définir comme ceci :

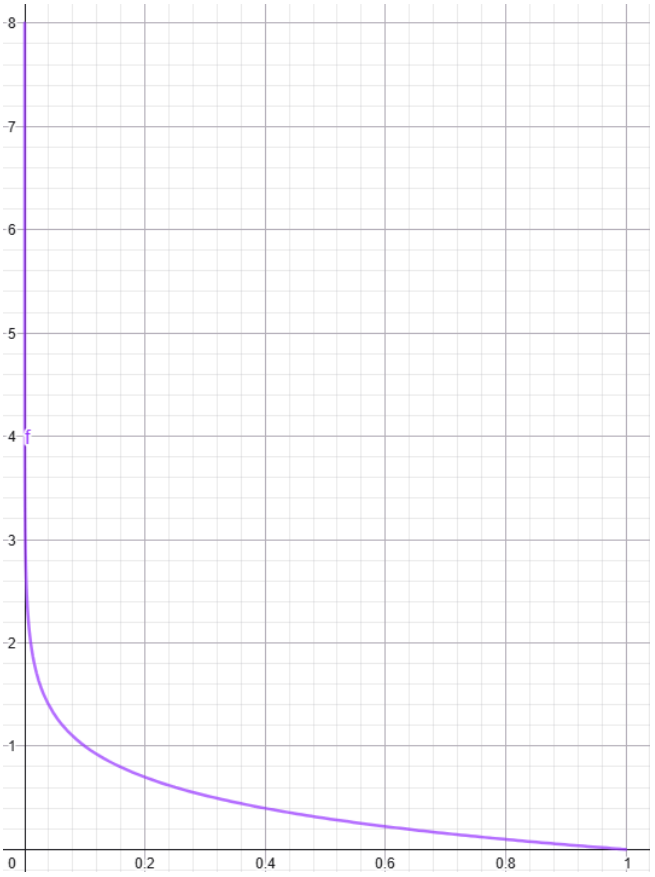
$$SoftMax(sorties) = \bigcup_{i=1}^{taille(sorties)} \{\frac{e^{sorties_i - max(sorties)}}{\sum_{k=1}^{taille(sorties)} e^{sorties_k - max(sorties)}}\}$$

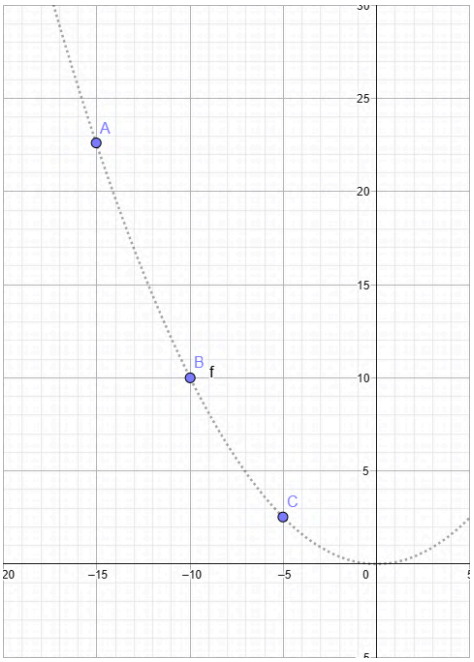
Cette fonction ramène donc toutes les sorties à une valeur entre 0 et 1 correspondante à la certitude à laquelle le réseau de neurones pense que l'entrée correspond à chaque sortie.

Ce sont donc ces éléments qui, une fois mis ensemble, forment le "forward path" de notre réseau de neurones.

## Le "backward path"

Le "backward path" correspond au chemin de rectification des erreurs de notre réseau de neurones et est donc fortement utilisé lors de la phase d'entraînement du réseau de neurone, mais pour ce faire, il faut tout d'abord pouvoir évaluer l'erreur de notre réseau de neurones. Pour ce faire, nous avons la "Categorical Cross-Entropy function" qui est définie comme ceci : $CCE(x) = -ln(x)$ où $x$ est la sortie de la classe attendue lors de la phase d'entraînement.






À partir de cela, il est possible d'entraîner le réseau de neurones, ce qui signifie donc trouver la valeur de chaque poids et chaque biais pour laquelle la perte est la plus faible dans toutes les situations déjà données par le dataset d'entraînement. Dans notre cas, le dataset qui sera utilisé sur le réseau de neurones final est celui donné par Kaggle, un site de dataset public. Pour corriger les erreurs de notre réseau de neurones, la méthode utilisée pour l'instant est de calculer la perte dans trois cas distincts : le premier est sans modification, le deuxième est l'ajout d'une valeur $shift$ au poids ou au biais que l'on essaie d'améliorer, le troisième est le retrait de cette valeur $shift$ au poids ou biais. Après le calcul de la perte, dans les trois cas, on garde la version du poids ou biais ayant eu la perte la plus faible. Une fois cette étape faite sur chaque poids et chaque biais du réseau, elle est réitérée avec une valeur $shift$ se rapprochant de 0 sans jamais l'atteindre ni la dépasser, après suffisamment d'itération, le réseau de neurones est assez précis pour être utilisé.

La méthode présentée ci-dessus est celle qui a été implémentée lors de la première soutenance. Cependant, nous avons était trop ambitieux par rapport à nos objectifs ce qui à fini par nous porter préjudice car en effet, la backpropagation à implémenter pour cette soutenance finale ne fonctionne pas dû à plusieurs problèmes que nous avons identifié seulement trop tard.

Sans ce problème, les autres parties de notre réseau final à 2 couches cachés fonctionne parfaitement, ainsi nous arrivons bien à passer des inputs à nos outputs cependant, ce qui nous pose un gros problème est le calcul des gradients, des poids et des biais. Nous n'avons pas réussi à identifier le problème c'est pourquoi, pour cette soutenance nous avons décidé d'avoir une fonction qui permet d'importer et d'exporter les différentes données de notre réseau de neurones.

La formule utilisée afin d'ajuster les poids et biais d'un réseau de neurone se présente de la façon suivante:

$$\Delta W_{kj}^{l} = -\epsilon \frac{\partial E}{\partial a_k^l} = -\epsilon \frac{\partial E}{\partial a_k^l} \frac{\partial a_k^l}{\partial z_k^l} \frac{\partial z_k^l}{\partial W_{kj}^l}$$

- Avec $W^{l}_{kj}$ le biais appartenant à la couche $l$ et à l'emplacement $(k, j)$
- $\varepsilon$ le ratio d'apprentissage (une valeure située entre 0 et 1)
- $E$ la perte de notre réseau de neurones
- $a^{l}_{k}$ la valeure de sortie du neurone situé à l'emplacement $k$ de la couche $l$ après la fonction d'activation
- $z^{l}_{k}$ la valeure de sortie du neurone situé à l'emplacement $k$ de la couche $l$ avant la fonction d'activation

Dans notre cas, le choix de notre fonction de perte et celle de fonction d'activation pour la dernière couche de notre réseau de neurones permet de simplifier certains calculs, en effet le calcul des nouveaux poids de la dernière couche de notre réseau de neurones se résume à:

$$\frac{\partial E}{\partial W_{kj}^L} = \frac{\partial E}{\partial z_k^L} \frac{\partial z_k^L}{\partial W_{kj}^L}$$

$$\delta_k^L = \frac{\partial E}{\partial z_k^L} = a_k^L - t_k$$

Avec $t_k$ égale à la sortie voulu par notre réseau de neurones

$$\frac{\partial E}{\partial W_{kj}^L} = \frac{\partial E}{\partial z_k^L} \frac{\partial z_k^L}{\partial W_{kj}^L} = \delta_k^L a_j^{L-1}$$

$$\frac{\partial E}{\partial b_k^L} = \frac{\partial E}{\partial z_k^L} \frac{\partial z_k^L}{\partial b_k^L} = \delta_k^L (1) = \delta_k^L$$

Toutefois ces formules ne peuvent pas être utilisés pour les autres couches de notre réseau de neurones car ces couches ci n'utilisent pas les mêmes fonctions d'activations mais aussi car pour calculer la dérivée partielle de la perte par rapport à un poids situé sur une couche cachée il faudra passer par la dérivée partielle de la parte par rapport à la sortie de chaque neurone de la couche d'après.

Similarly, Equation (8) - (10) derives each component of Equation (7)

$$\frac{\partial E}{\partial W_{ji}^{l-1}} = \frac{\partial E}{\partial a_j^{l-1}} \frac{\partial a_j^{l-1}}{\partial z_j^{l-1}} \frac{\partial z_j^{l-1}}{\partial W_{ji}^{l-1}} \tag{7}$$

$$\begin{aligned} \frac{\partial E}{\partial a_j^{l-1}} &= \sum_k \frac{\partial E}{\partial z_k^l} \frac{\partial z_k^l}{\partial a_j^{l-1}} \\ &= \sum_k \delta_k^l W_{kj}^l \end{aligned} \tag{8}$$

$$\frac{\partial a_j^{l-1}}{\partial z_j^{l-1}} = f'(z_j^{l-1}) \tag{9}$$

$$\frac{\partial z_j^{l-1}}{\partial W_{ji}^{l-1}} = a_i^{l-2} \tag{10}$$

Combining all together, we get

$$\frac{\partial E}{\partial W_{ji}^{l-1}} = a_i^{l-2} f'(z_j^{l-1}) \sum_k \delta_k^l W_{kj}^l. \tag{11}$$

We can define $\delta_j^{l-1} = \frac{\partial E}{\partial a_j^{l-1}} \frac{\partial a_j^{l-1}}{\partial z_j^{l-1}} = f'(z_j^{l-1}) \sum_k \delta_k^l W_{kj}^l$

and re-write Equation (11)

$$\frac{\partial E}{\partial W_{ji}^{l-1}} = \delta_j^{l-1} a_i^{l-2}. \tag{12}$$

- Avec $f'(x)$ étant la dérivée de la fonction

Ici nous avons traité uniquement le cas des formules pour les poids, mais il faut aussi prendre en compte les formules pour les biais. (pour plus de détails voir bibliographie).

De part leur nature compliquée ces fonctions ont particulièrement été compliqués à implémenter en C dans notre projet. En effet la fonction permettant la correction des biais et poids de notre réseau de neurones à nécessité pas moins de 6 versions différentes avant d'avoir celle présente actuellement.

Malheureusement la rétropropagation ne marche pas comme nous l'espérons, en effet au fur et à mesure du développement, nous avons remarqué 2 effets secondaires indésirables:

Le premiers est que dans certains cas (en fonction de la taille des couches cachés ou de notre *learning rate*) pendant l'entraînement, une des valeurs brute de sortie de notre réseau de neurones sera bien plus grande (en valeur absolue) que la normale, ce qui à pour conséquence de transmettre des poids beaucoup trop élevé comparé aux autres ce qui ne permet pas d'avoir un résultat lisse et qui avantage grandement les premières lettres présentes dans notre ensemble d'entraînement.

Par la même occasion, pour palier à ce problème, nous avons décidé de créer une backpropagation qui prend des lettres aléatoires et qui ne va pas d'abord s'entraîner avec tous les A, puis tous les B, etc…

Le second problème est lors du calcul de nos différents gradients. Malgré des tentatives de debug avec GDB, nous savons qu'un problème persiste dans cette partie de notre code et malgré la réécriture de ce dernier cela ne marche toujours pas.

Nous sommes profondément déçu de nous dû à ce problème qui aura persisté jusqu'au rendu du projet de plus que malgré des heures à plusieurs à être passé dessus nous n'avons toujours pas trouvé la source de ces erreurs.

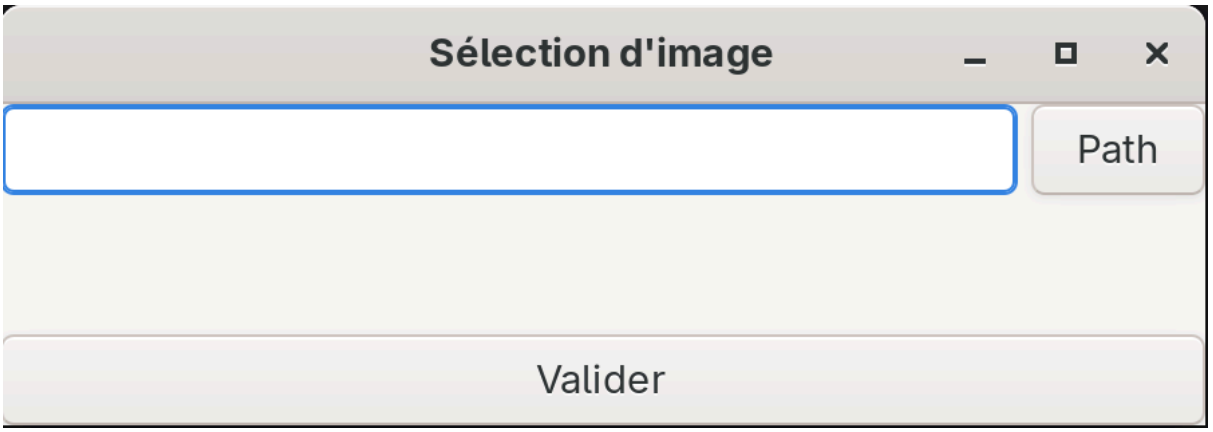
# L’interface graphique

La principale difficulté de l’interface graphique est de lier les différentes parties quelque chose de cohérent, de clair et de fluide. Pour ce faire, nous avons choisi comme GUI toolkit gtk, qui est particulièrement avantageux pour créer une interface graphique en C, car il est nativement écrit en C, sans couche d’abstraction lourde, ce qui offre de bonnes performances et un contrôle fin du comportement de l’UI. Il fournit un large ensemble de widgets, et des layouts adaptatifs, tout en restant portable. Sa licence LGPL permet une utilisation aussi bien libre que commerciale.

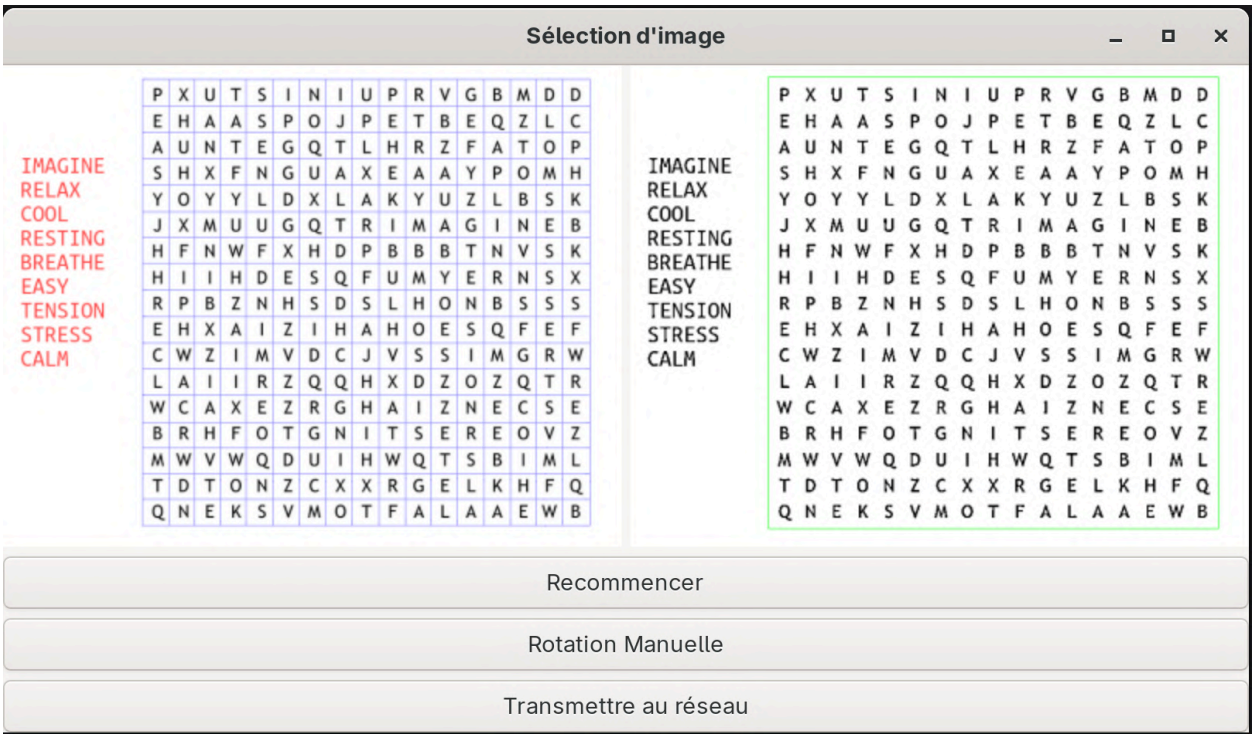
Le premiers est que dans certains cas (en fonction de la taille des couches cachés ou de notre learning rate) pendant l'entraînement, une des va

## Chargement de l’image et pre processing

Au lancement du programme, l’ui nous demande le lien vers la photo du mot mêlée à résoudre.




Une fois le chemin récupéré, on charge l’image avec SDL, puis on le transforme en notre type Image, on applique le pre process, puis on exporte l’image et on affiche à gauche l’image de départ et à droite l’image après le pre process. En cas d’erreur de la rotation automatique, on a laissé la possibilité à l’utilisateur de rentrer manuellement la rotation. On a également laissé la possibilité de charger une autre image.




## Transfert au réseau de neurone

Afin de pouvoir résoudre notre grille, nous devons tout d'abord reconstruire cette dernière ainsi que la liste de mots à chercher. Par exemple pour reconstruire notre grille nous allons tout d'abord récupérer les dimensions de la grille car au dossier exporter par le pré-processing et qui est sous la forme :

{nom_de_l'image}_{nombres_de_lignes}_{nombres_de_colonnes}

et les cellules de la grille contiennent simplement l'indice de leur abscisse et de leur ordonnée. Grâce à ça, nous créons un nouveau fichier qui a le même nom que l'image correspondante dans lequel nous allons écrire la grille. Nous faisons donc une double boucle qui permet d'accéder à toutes nos cellules et pour chacune de ces cellules nous envoyons leur correspondance dans notre réseau de neurones préalablement entraîné. Ensuite, nous ajoutons la lettre obtenue au fichier créé et ainsi de suite jusqu'à obtenir l'intégralité de notre grille.

Pour la liste de mots, la méthode reste très similaire avec un nouveau fichier créé et les lettres qui passent une par une par notre réseau de neurones pour ressortir les unes à la suite des autres afin d'être transmis au solveur.

## Transfert au solver

Une fois la liste de mot et la grille validé, chaque mot et envoyé au solveur les coordonnées de la matrice sont ensuite convertis en coordonné de l'image et un trait est tracé via SDL entre le point de départ et d'arrivée. Une fois la liste de mot terminée, l'image est affichée.

# Bibliographie

Site permettant de comprendre les réseaux de neuronnes :
http://neuralnetworksanddeeplearning.com/index.html

Source utilisé pour créer le réseau de neurones :

https://victorzhou.com/blog/intro-to-neural-networks/

Les détails sur la fonction d'activation Softmax :

https://en.wikipedia.org/wiki/Softmax_function

Dataset :
https://www.kaggle.com/datasets/preatcher/standard-ocr-dataset

Librairie SDL :
https://www.libsdl.org/

Pour comprendre la transformée de Hough :

https://fr.wikipedia.org/wiki/Transform%C3%A9e_de_Hough

Pour comprendre la rétropropagation:

https://doug919.github.io/notes-on-backpropagation-with-cross-entropy/