VOX MACHINA :
ANNA

Rapport de Soutenance 1

Ruben Gervais
Ilyann Gwinner
Nathan Hirth
Josselin Priet

Table des matières

1 Introduction

1.1 Présentation des membres

— Ruben Gervais : Après un Bac général en mathématiques et physique-chimie, je porte beaucoup d'intérêt sur les sciences et en particulier l'informatique. J'ai appris à programmer en python et java sans projet précis en objectif. Je suis dirigé par une grande curiosité et une envie de dépassement de soi. C'est pourquoi ce projet est pour moi une chance et un défi à relever. En apportant mes qualités au groupe, j'espère l'aider à se dépasser et à obtenir des résultats dont nous serons fiers.

— Ilyann Gwinner : Je me suis passionné pour l'informatique en fin 5ème quand un professeur de physique-chimie a parlé d'un langage nommé python. Depuis ce jour, j'ai appris de nombreux langages. J'ai également découvert la cybersécurité qui devint assez rapidement mon centre d'intérêt principal. Depuis quelques années, avec l'explosion de l'IA, j'ai commencé à énormément m'y intéresser. Aujourd'hui, je m'intéresse à beaucoup de domaine de l'informatique comme le hardware, le développement ou encore l'ordinateur quantique, mais la cybersécurité et l'IA reste ce que je préfère. C'est pour cela que je suis heureux de faire ce projet qui me permet de m'en apprendre plus sur l'IA et j'espère comprendre correctement son fonctionnement avec la fin du projet.

— Nathan Hirth : J'ai commencé à m'intéresser grandement l'informatique, il y a un peu plus de six ans. Durant ces quelques année, j'ai pu découvrir différent domaine de l'informatique tel que la création de jeux vidéo, la création de site web, la cybersécurité... Mais je n'ai jamais essayé de faire de l'IA en profondeur malgré le fait que c'est l'un des domaines qui m'intéresse le plus avec celui de la cybersécurité. J'ai réalisé quelque IA, mais avec l'aide de bibliothèque déjà faite, c'est pour cela que pour ce projet, j'ai souhaité me pencher sur la réalisation de l'IA pour la reconnaissance de caractères. Comprendre comment fonctionne l'IA, quelles sont les fonctions qu'elle utilise pour obtenir de tels résultats, c'est ce que j'aimerais atteindre à la fin de ce projet.

— Josselin Priet : Intéressé à l'origine par le journalisme, je me suis très vite redirigé vers l'informatique durant la période du lycée. J'ai ainsi pu peaufiner mes connaissances et découvrir davantage ce monde, ce qui m'a permis d'engranger de l'expérience. Ce projet est donc pour moi une nouvelle manière d'apprendre dans un nouveau langage. Malgré cela, j'espère aussi pouvoir utiliser les outils et astuces que j'ai pu développer depuis le lycée, en particulier en algorithmie.

1.2 Tâches

Tâche	Avancement prévu	Avancement réel	Responsable
Preprocessing			/
GrayScale	100%	100%	Ruben
Binarisation	100%	100%	Ruben
Rotation Manuelle	100%	100%	Josselin
Rotation Automatique	0%	50%	Josselin
Cropping	100%	70%	Ruben
AI			/
NXOR	100%	100%	Ilyann et Nathan
ANNA	50%	70%	Ilyann et Nathan
Other			/
Solver	100%	100%	Nathan
UI	60%	60%	Ilyann

On peut voir que les objectifs pour cette soutenance sont plutôt bien respectés et même dépasser sur certain point. Mais nous pouvons aussi voir un retard en ce qui concerne le cropping. En effet, il ne marche pas totalement, certaines lettres son mal détectées et le découpage de ses dernières n'est pas très bon à certain endroit. Ce retard peut être expliqué par une mauvaise gestion du temps et peu être aussi de la répartition des taches. En effet, nous avons pris de l'avance sur l'IA et la rotation alors que le cropping n'était pas finalisé. C'est donc dans un rush final que nous avons dû nous pencher activement sur le cropping.

2 Preprocessing

Cette section contient tout ce qui se passe avant l'AI. Pour le préprocessing, l'image passera par une étape de grayscale, de binarisation, de rotation et enfin de cropping.

2.1 GrayScale

Le grayscale est une étape importante du préprocessing. Cette étape permet de passer une image originellement en couleur en niveau de gris. Pour ce faire, il existe une formule qui utilise les valeurs RGB de l'image qu'on cherche à convertir. Voici la formule que nous avons utilisée :

$$GrayScale = 0.3 * R + 0.59 * G + 0.11 * B$$

Où R est le niveau de rouge du pixel, G le niveau de vert et B le niveau de bleu.

Cette conversion permet de travailler non plus avec trois valeurs par pixel, mais plus qu'une seule valeur.

Voici un exemple de ce que donne le grayscale :

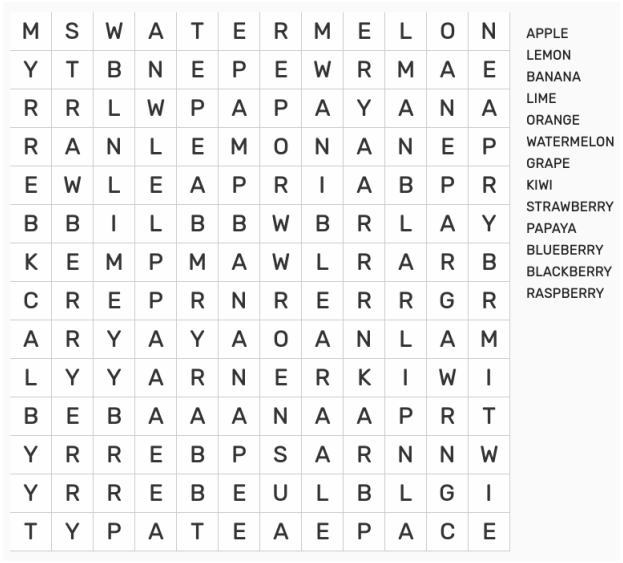

FIGURE 1 – Application du grayscale

2.2 Binarisation

Maintenant que nous sommes passés de trois valeurs par pixel à une seule valeur, il nous faut passer l'image en noir et blanc sans nuances de gris. Cela permet de simplifier l'image en éliminant les détails inutiles.

Pour effectuer cette binarisation, nous utilisons l'algorithme d'Otsu qui est une méthode de binarisation de l'image qui vise à séparer les pixels d'une image en deux classes distinctes : le premier plan (les objets d'intérêt) et l'arrière-plan.

Cet algorithme contient plusieurs étapes :

- Tous d'abord, on commence par calculer l'histogramme des niveaux de gris de l'image, ce qui permet d'obtenir la distribution des intensités de gris.

- Ensuite, nous avons le calcul du seuil optimal. Pour ce faire, il existe une formule que voici :

$$\sigma_\omega^2(t) = \omega_1(t)\sigma_1^2(t) + \omega_2(t)\sigma_2^2(t)$$

$$\sigma_b^2(t) = \sigma^2 - \sigma_\omega^2(t) = \omega_1(t)\omega_2(t)\left[\mu_1(t) - \mu_2(t)\right]^2$$

Où les poids ω_i représentent la probabilité qu'un pixel appartienne à la classe i, chaque classe étant définie par un seuil t. Et où les σ_i^2 sont les

variances de ces classes.

- Enfin, il nous reste plus qu'à appliquer ce seuil. Pour ce faire, on vérifie pour chaque pixel s'il est supérieur au seuil, si c'est le cas, on change la valeur de ce pixel à 0 sinon on la met à 255.

Voici un exemple d'image après avoir appliqué la binarisation :

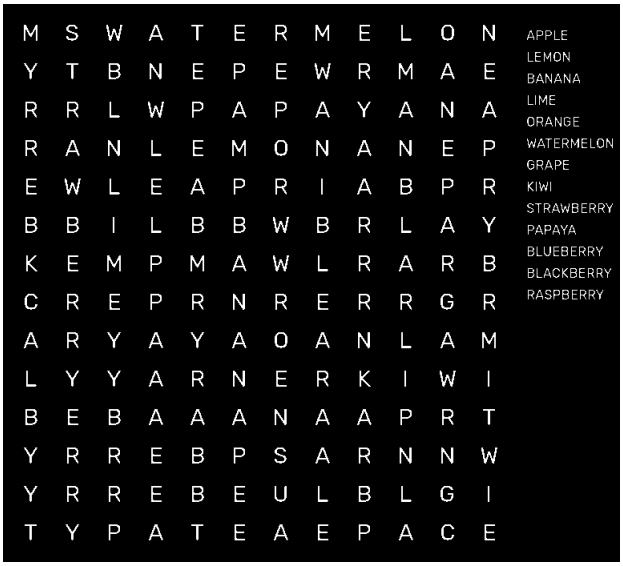

FIGURE 2 – Application de la binarisation

2.3 Rotation

2.3.1 Rotation Manuelle

L'objectif est de faire tourner une image en spécifiant un angle en degrés et de l'afficher ensuite grâce à SDL.

Fonctionnement :

En utilisant les fonctions de base de SDL et la bibliothèque mathématique pour convertir entre degrés et radians, le programme parvient à effectuer facilement des rotations. SDL fournit des outils simples pour gérer les transformations, ce qui facilite la mise en œuvre de la rotation manuelle.

2.3.2 Rotation Automatique

Le deuxième objectif, plus complexe, est de faire tourner l'image automatiquement, sans spécifier d'angle. Le programme doit ainsi détecter seul si l'image n'est pas droite et appliquer la rotation nécessaire. Actuellement, la fonction de rotation manuelle fonctionne parfaitement, tandis que la rotation automatique n'est concluante qu'une fois sur

deux.

Fonctionnement :

La rotation automatique utilise des techniques de binarisation et de conversion en niveaux de gris, développées par Ruben, pour obtenir une image plus nette. Cela simplifie le traitement de l'image, puis, en appliquant plusieurs algorithmes mathématiques, dont les algorithmes de Sobel et de Canny, le programme parvient à détecter les contours et à déterminer l'angle de rotation. Pour le moment, cette partie du code reste imparfaite, la rotation automatique fonctionnant de manière fiable uniquement sur un petit échantillon d'images.

Échantillons et tests :

Le programme a été testé sur un grand nombre d'images, ce qui a permis d'identifier certains problèmes, comme le fait que la rotation automatique fonctionne correctement sur un petit nombre d'échantillons, mais devient défectueuse sur un plus grand ensemble. Tous les tests ont été réalisés manuellement, sans interface graphique, en utilisant des Makefiles et gcc.

2.4 Cropping

On veut d'abord chercher à faire la différence entre les mots de la liste et la grille de lettres. Pour ca, on va utiliser un algorithme de détection des composantes connectées. Une fois ces composantes sélectionnées, nous les séparons entre mots de la liste et lettre de la grille grâce à leur rapport largeur/hauteur.

Détection des Composantes Connectées :

Un algorithme de "flood-fill" (ou remplissage par diffusion) est utilisé pour détecter les composantes connectées. Cette méthode permet d'identifier des zones contiguës dans l'image en regroupant les pixels connectés de même couleur (typiquement les pixels blancs représentant les lettres).
Chaque composante connectée est étiquetée avec un identifiant unique pour permettre un traitement ultérieur.

Calcul des Boîtes Englobantes :

Pour chaque composante connectée, une boîte englobante (bounding box) est calculée. Cela implique de déterminer les coordonnées minimales et maximales en x et y des pixels de la composante.
À partir de ces coordonnées, la largeur et la hauteur de la boîte sont

calculées.

Séparation des Mots et des Lettres Basée sur le Rapport d'Aspect (Aspect Ratio) :

Pour chaque boîte englobante, le rapport largeur/hauteur est calculé. Un mot (généralement une séquence de lettres dans la liste) aura un rapport largeur/hauteur significativement plus élevé qu'une lettre unique dans la grille.
Une règle empirique est utilisée pour déterminer ce seuil. Par exemple :
- Si le rapport largeur/hauteur est supérieur à 2, il s'agit probablement d'un mot de la liste.
- Si le rapport est proche de 1 (lettre presque carrée), il s'agit probablement d'une lettre de la grille.

Voici un exemple de l'image après la séparation :

FIGURE 3 – Séparation des Mots et des Lettres

Extraction et Enregistrement des Composantes :

Pour les mots de la liste :

Chaque mot est extrait en tant qu'image individuelle. Pour chaque mot, un sous-algorithme sépare les lettres individuellement en utilisant des sous-boîtes englobantes ou une détection de contours.

Pour les lettres de la grille :

Chaque lettre est extraite et redimensionnée à une taille fixe, par exemple 32x32 pixels, pour uniformiser les données. Le nommage des

lettres de la grille suit un format spécifique (par exemple, grid_{x_index}_{y_index}.png), en utilisant leur position dans la grille.

3 AI

Cette section se divise en deux sous-parties. La première est une AI qui a pour but de reconnaitre la porte logique NXOR ($A.B + \overline{A}.\overline{B}$). Ce modèle permet de vérifier si l'architecture du réseau créé pour la deuxième partie est capable d'apprendre sur un système simple. La seconde est une AI capable de reconnaitre les lettres de l'alphabet minuscule et majuscule. Nous avons name cette deuxième IA ANNA pour Advanced Neural Network Analyser.

3.1 NXOR

Pour réaliser l'IA qui reconnait la porte NXOR, nous avons utilisé un modèle simple de réseau de neurone. Cette IA contient deux neurones d'entrer, trois neurones dans la couche cachée et un neurone de sortie. Il contient donc trois couches de neurones.

Afin de fonctionner, ces différents neurones sont reliés par des synapses. Ces synapses relient les neurones d'entrer aux neurones de la couche cachée et les neurones de la couche cachée au neurone de sortie. Chaque synapse contient deux valeurs pour pouvoir faire le lien entre les différentes couches, la première valeur est le poids de la synapse et la seconde est le biais.

Grâce à trois fonctions permettant de calculer la sortie, et de calculer et de mettre en place l'erreur et la correction du modèle, l'IA à partir de poids et biais aléatoire peut apprendre à reconnaitre la porte NXOR.

- Forward Propagation : Cette fonction permet de calculer la valeur de la sortie. Elle multiplie les valeurs de chaque neurone de chaque couche avec les poids des synapses correspondant et en y ajoutant les biais pour donner les valeurs des neurones de la couche suivante. Une fois calculé, on passe le résultat de chaque neurone dans une fonction d'activation qui permet de garder la valeur entre 0 et 1, ici la fonction sigmoïde dont la formule est :

$$Sigmode = \frac{1}{1 + e^{-x}}$$

À la fin, la valeur du neurone de sortie nous donne la probabilité que le résultat du NXOR soit 1. Si cette probabilité inférieure à 0.5 alors la valeur de sortie est 0 sinon c'est 1.

- Back Propagation : La rétropropagation est une fonction qui permet

de calculer les erreurs du réseau de neurone afin de l'ajuster les résultats de l'IA. Cette fonction utilise la dériver de la fonction d'activation, donc pour la sigmoïde, nous avons :

$$SigmodePrime = x * (1 - x)$$

Une fois cette fonction appliquée sur les différentes couches du réseau, nous obtenons les valeurs pour l'ajustement des différents poids et biais de l'IA.

- Update : Cette fonction permet de mettre à jour les poids et les biais des différentes synapses. Nous utilisons la régulation L2 afin d'optimiser la rapidité d'apprentissage du réseau. Cette fonction va aussi utiliser une variable qu'on appelle learning rate. Cette variable permet de contrôle de la vitesse de convergence, de stabilité de l'apprentissage et d'équilibre entre exploration et convergence. Plus elle est petite, plus le modèle convergera, mais le temps d'apprentissage sera plus élevé.

Pour vérifier et suivre l'apprentissage de l'IA, nous utilisons une loss function du nom de Log Loss dont la formule est :

$$LogLoss = -\frac{1}{N}\sum_{i=1}^{N}(y_i * log(p_i) + (1 - y_i) * log(1 - p_i))$$

Avec N le nombre d'observations, yi le résultat binaire réel (0 ou 1) pour la i-ième observation et pi la probabilité prédite que la i-ième observation.
Cette fonction permet d'avoir la valeur de l'écart entre les résultats du modèle et donc sa performance. Si l'IA donne un résultat de 1 à 60% et un 1 à 90%, la fonction Log Loss donnera un résultat plus petit pour le deuxième cas.

Afin d'optimiser l'apprentissage du réseau, nous initialisons les poids et les biais aléatoirement grâce à l'initialisation de Xavier qui est faite pour la fonction d'activation sigmoïde.

Une fois le programme lancer, le modèle arrive à 100% de réussite en environ 300 boucle d'entraînement (chaque boucle comprendre les trois fonctions vu précédemment).

3.2 ANNA

3.2.1 Neural Network

Contrairement au NXOR le réseau de neurone artificiel ANNA que nous utilisons pour reconnaitre des lettres, possède quatre couches de neurones au lieu de trois. Sa couche d'entrer contient 256 neurones, ce qui correspond à des images de 16x16 pixels. Les couches cachées

contiennent dans les derniers essais 512 neurones pour la première et 256 pour la deuxième. Et sur la couche de sortie, nous avons 26 neurones correspondant au 26 lettre de l'alphabet.

Le fonctionnement d'ANNA est similaire à celui du NXOR. Pour fonctionner, nous avons toujours les trois fonctions principales : Forward Propagation, Back Propagation et Update. La fonction d'activation que nous avons choisie pour cette IA est la fonction ReLU dont la formule est :

$$ReLU = max(0, x)$$

et sa dériver est :

$$ReLUPrime(x > 0) = 1$$

$$ReLUPrime(x <= 0) = 0$$

Afin d'initialiser correctement les poids du modèle avec la fonction d'activation ReLU, nous utilisons l'initialisation de He.

Les valeurs d'entrer d'ANNA sont quelque peu différentes des valeurs du NXOR. En effet, nous prenons en entrer les valeurs de chaque pixel d'une image en escale donc entre 0 et 255 puis nous divisions ses valeurs par 255 afin d'obtenir des valeurs entre 0 et 1 en entrer du modèle.

Si nous utilisions le réseau de neurone seulement avec les modifications du NXOR expliquer précédemment, cela ne peut pas marcher. En effet, la différence de complexité pour reconnaitre une porte NXOR et une lettre sur une image est énorme. C'est pour cela que nous allons ajouter plusieurs choses à notre modèle :

- Le dropout est une technique de régularisation utilisée pour améliorer la généralisation d'un modèle en réduisant le surapprentissage (overfitting). Elle consiste ne le fait de désactiver certain neurone au moment de la Forward Propagation.

- L'optimiseur Adam (pour "Adaptive Moment Estimation") est un des algorithmes d'optimisation les plus populaires en apprentissage profond. Il combine les avantages de deux autres méthodes, AdaGrad et RMSProp. Cet algorithme vient replacer la fonction Update du NXOR. Il permet d'ajuster le learning rate vu précédemment pour chaque paramètre du modèle (les poids et les biais). Cela permet d'augmenter considérablement l'efficacité de l'entrainement du modèle.

- Une fonction qui permet de mélanger le dataset qui s'exécute à chaque boucle. Cette fonction est très importante pour éviter que l'IA apprenne juste l'ordre des réponses.

- Pour finir, le Softmax qui est une fonction mathématique permettant de transformer les sorties d'un modèle en une distribution de probabilité. Cette fonction permet d'amplifier les différences entre les différentes valeurs. Si une valeur est élevée, elle le sera encore plus et si une valeur est basse, elle le sera aussi encore plus.

Pour avoir le résultat d'une image quelconque, nous avons juste à utiliser la fonction de Forward Propagation avec les valeurs des poids et des biais déjà entrainer. C'est pour cela qu'à chaque boucle d'entraînement de notre AI, nous enregistrons les différentes valeurs et poids et biais de notre modèle des fichiers csv. Puis quand on reprend l'entrainement ou quand on utilise le modèle pour prédire la lettre sur une image, nous chargeons ces fichiers csv.

3.2.2 Dataset

Entrainer un modèle d'IA tel qu'ANNA demande un certain nombre de données qui doit être judicieusement choisi afin d'être le plus efficace possible. Dans notre cas, nous utilisons des images de lettre que l'on a généré grâce aux nombreuses police d'écriture de Google Fonts. Nous avons par la suite classé ces différentes images par caractère dans des dossiers portant le nom de ces dits caractères

Après avoir récupéré les différentes images de lettre, nous avons créé un programme qui a pour but de récupérer les pixels des images lettre par lettre et de les mettre dans plusieurs fichiers csv. Ces fichiers comportent chacun toutes les lettres répéter le même nombre de fois, mais avec des polices d'écriture différentes. Pour chaque fichier csv créé, on en crée un deuxième qui contient le résultat que l'IA doit donner. Ces différents fichiers permettent d'entrainer l'IA sur de plus petit lot de donner afin de vérifier que notre modèle marche bien sans avoir à attendre trop longtemps. Ces fichiers correspondent au train set. On a aussi créé un fichier supplémentaire avec plus de lettre et un fichier pour ses résultats. Ce fichier a pour but d'avoir des données sur lequel l'IA ne s'est pas entrainé afin de vérifier son efficacité sur de nouvelle donnée. Ce fichier correspond au test set.

À la suite de ces opérations, nous avons obtenu 23 fichiers de 13000 lettres dans le train set ce qui permet de mettre à jour les poids 23 fois en une boucle et un fichier de 20748 lettres dans le test set. Nous avons entrainé notre modèle de longues heures sur ce dataset en changeant certain paramètre et en enregistrant dans un nouveau fichier csv les différents résultats du modèle afin de faire un graphe pour chaque configuration pour trouver la meilleure. Nous sommes finalement arrivés à un plafond d'apprentissage avec la configuration expliqué au-dessus.

Voici les résultats obtenus après une centaine de boucles :

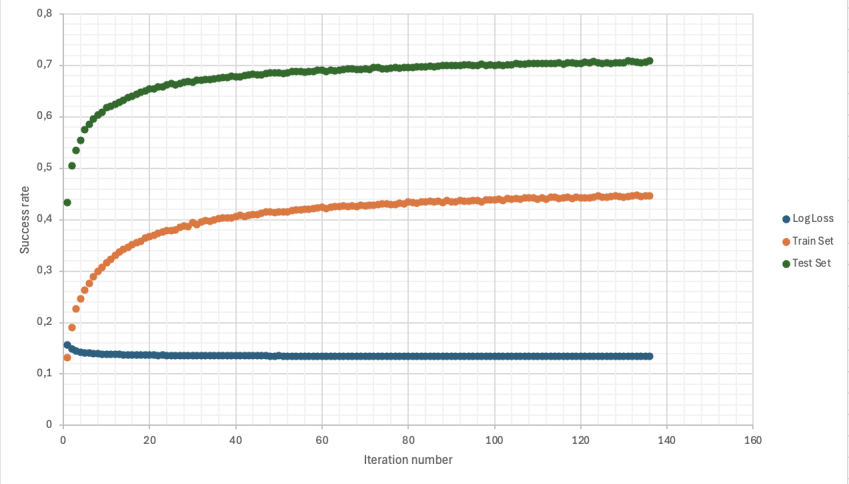

FIGURE 4 – Statistique du modèle sur 136 boucles

Nous pouvons voir sur ce graphique que le pourcentage de réussite du train set commence autour de 0,15 (15%) et augmente pour atteindre environ 0,45 (45%) avant de se stabiliser. La courbe du test set suit une trajectoire similaire à celle du train set, mais elle démarre plus haut et se stabilise autour de 0,7 (70%). On voit aussi que la courbe du LogLoss reste quasiment constante autour de 0,1 tout au long des itérations. Un LogLoss aussi constant suggère que le modèle n'est pas en train d'apprendre efficacement.

3.2.3 Problèmes rencontrés

Avant d'arriver à la configuration actuelle de notre IA, nous avons testé plein d'architecture différente, mais aucune d'entre elle ne passer les 10% de réussite dans le train set. Mais même après toutes ces recherches et changements, nous arrivons seulement à 45% de réussite sur le train set et 70% sur le test set. Et surtout, après un certain nombre de boucles, le modèle arrête d'apprendre. Et avec le LogLoss, on voit que le modèle n'est pas en train d'apprendre efficacement.
Nous avons plusieurs théories pour expliquer ces résultats. La première est que le dataset n'est pas adaptée ou contient trop d'élément. La seconde est que notre architecture d'IA n'est pas adaptée à de la reconnaissance de caractère.

3.2.4 Optimisation

Entrainer une telle IA peu prendre un certain temps, c'est pour cela que nous avons cherché des moyens d'optimiser la vitesse d'apprentissage. Pour cela, nous avons eu l'idée de faire certaine partie de l'entraînement en multithreading grâce à la bibliothèque *pthread*. Nous l'avons

notamment utilisé dans les calcule matriciels présents partout dans l'entrainement de l'IA. Cela nous a permis de réduire considérablement le temps d'entraimer.

3.2.5 CNN

Afin de résoudre les problèmes cités précédemment, nous nous sommes renseignés sur des architectures faites exprès pour la reconnaissance d'image.

Le Convolutional Neural Network (CNN) est une architecture permettant de classer différente image. Son principe est simple, il applique différent filtre à l'image donnée en entrer puis il donne le résultat de ces filtres au réseau de neurone. Ces filtres ont pour but de mettre en évidence certaine partie spécifique de l'image afin que le modèle se focalise dessus.
Il existe deux types de filtre :

- Les filtres de convolution permettent de rendre l'image plus nette et de faire ressortir les lignes et les colonnes.

- Les filtres de pooling permettent de flouter l'image en prenant la moyenne ou le maximum d'une zone.

Une fois ces filtres appliquée à l'image, sa taille sera réduite ce qui réduit le temps d'apprentissage. Il existe déjà plusieurs architectures connues utilisant le CNN. Pour notre par, nous pensons utiliser le LeNet-5.

4 Other

Cette dernière section explique toutes les fonctions utilisées après l'application de l'AI aux différentes lettres. Elle qui contient le solver ainsi que l'UI qui regroupe tous les codes.

4.1 Solver

Le solver est le programme permettant de résoudre la grille de mots cachés. Il prend en entrer un fichier contenant une grille de lettre qui correspond à la grille dans laquelle l'on doit retrouver les mots. Cette fonction prend deux paramètres : le premier est celui du fichier de la grille et le deuxième est le mot à retrouver dans la grille.

Afin de trouver le mot à chercher, on récupère les donner du fichier contenant dans la grille et on le stocke dans un tableau. Pour chaque lettre du tableau, on regarde si elle correspond à la première lettre du

mot à chercher et si c'est le cas, on prend le nombre de lettres qu'il reste dans chaque direction et on le compare à la longueur du mot pour voir s'il peut être dans cette direction. Puis pour chaque direction valide, on regarde la deuxième puis troisième lettre... et on les compare aux lettres du mot, s'il y a une différence, on arrête, sinon ont continu. Si on trouve le mot, on renvoie la position de départ ainsi que celle d'arriver, sinon on renvoie "Not Found".

Pour d'optimiser la recherche du mot, nous avons pensé à parcourir dans les deux sens en même temps quand on cherche dans une direction en vérifiant le mot à l'envers. Cela nous permet de ne parcourir que la moitié de la grille en lettre par lettre.

4.2 UI

Comme dit plus tôt, l'UI est la partie visuelle de l'application, mais c'est également cette partie qui va appeler les différents codes un par un. Pour l'instant, nous nous sommes concentrés sur la partie programme principal plus que sur la partie design.

Le programme a plusieurs étapes et fonctionnalités. Tout d'abord, il faut choisir une image, ensuite, il faut choisir si on veut le mode pas à pas ou non. Le mode pas à pas permet de voir chaque étape (GrayScale, Binarisation...) et si on ne l'active pas, nous avons directement le résultat.

5 Conclusion

Pour conclure, nous sommes assez fiers de notre avancement pour cette première soutenance. Malgré un certain retard, notamment au niveau de préprocessing quelques jours avant le rendu, ce qui a conduit à un rush de dernière minute, nous avons réussi à finir la plupart des choses demandées pour cette soutenance.

Nous sommes plutôt fiers du résultat de notre IA sachant tout ce qu'on a dû tester avant d'en arriver au résultat actuel. Le NXOR montre de très bon résultat sur les testes que nous avons effectué. Et notre réseau de neurone permettant de reconnaitre les lettres montre lui aussi des résultats encouragent avec un tôt de réussite de 45% en une centaine de boucles et sur à peu près 300000 images.

Mais cette avance sur l'IA, mais également sur la rotation automatique qui est presque entièrement fonctionnelle, nous a fait prendre du retard sur le cropping qui contient encore quelques problèmes de détection et distinction entre les lettres et les mots ainsi que dans le

découpage de ces lettres.

Nous avons pris en compte l'erreur que nous avons commise dans la répartition des taches, ce qui a conduit à ce retard sur le cropping, pour ne pas la reproduire pour la dernière soutenance et pour avoir un projet fini et entièrement fonctionnel.