Socrate : Rapport de Projet

Purple Hat

Rapport de soutenance 2018

Merotto Enzo
Ajili Wassim
Zeitoun Jeremie
Zammit Baptiste

Avant de commencer les hostilités, nous allons retracer ensemble l'histoire de la création de ce second groupe au sein d'EPITA, pour réaliser des projets d'envergures. Nous avions commencé avec un jeu vidéo en 3D et multijoueur, et cette année nous sommes parti sur un OCR, un reconnaisseur de caractères pour être impolie. Notre groupe appelé "Purple Hat" n'est en fait que la concaténation de deux groupes de SUP, Palikorn et Quantum Arts. Les deux groupes ayant fusionné par lien d'amitié et par choix de pays, nous et vous sommes maintenant fin prêt pour écrire cette fabuleuse histoire.

Table des matières

Chapitre 1

Introduction

1.1. Description générale du rapport

Dans ce rapport, vous aurez la possibilité de suivre l'avancement du projet du troisième semestre dans son intégralité. Vous pourrez ainsi juger la qualité de notre travail et de notre persévérance dans l'adversité. Le rapport contient les différentes étapes de la construction de notre projet qui est un OCR nommé "Socrate", (OPTICAL CHARACTER RECOGNITION) pour cette première soutenance.
Pour suivre notre avancée, il y aura deux rubriques majeures qui composent notre OCR, *Traitement d'image* et *Réseau neuronal*.

1.2. Présentation des membres

1.2.1 Enzo *Louzo9818* Merotto

Login : *enzo.merotto*

Le projet me parait très intéressant. Il permet de bien comprendre le fonctionnement du traitement de texte en C ainsi que les prémisses du machine learning. C'est ce genre de projet qui m'a donne envie de faire de l'informatique. Mon premier projet informatique fut codé en python pendant mon année de lycée avec la spécialité ISN. J'ai donc par la suite décidé d'intégrer EPITA. Ma première année a l'EPITA a bien confirme que l'informatique me correspond. Cette année je commence à apprendre le langage de programmation C, et je l'apprécie même s'il me semble bien plus complique que les autres langages que je connais.

1.2.2 Wassim *Userleef* Ajili

Login : *wassim.ajili*

J'ai découvert l'informatique il y a plusieurs années et je me suis découvert un réel intérêt pour ce domaine, en particulier pour la programmation. J'ai donc appris à programmer en suivant des cours sur le web dans des langages comme le C ou le C++ (uniquement les bases) sur PC ou le TI-Basic sur une calculatrice. Mais c'est au lycée que mes connaissances dans ce domaine ont réellement augmenté. En effet, j'ai choisi d'intégrer la spécialisation Informatique et Science du Numérique qui m'a permis d'apprendre de nombreux langages comme l'HTML, le CSS, le Javascript

et surtout le Python. J'ai également pu réaliser plusieurs projets informatiques seul ou à plusieurs. Ce deuxième projet me permettra donc d'en apprendre encore plus et obtenir davantage d'expériences.

1.2.3 Jérémie *Kaijo* Zeitoun

Login : *jeremie.zeitoun*

Depuis tout petit j'ai beaucoup utilisé tout ce qui touche à l'informatique. J'ai toujours été très curieux de savoir comment les applications, utilitaire, etc. que j'utilise fonctionnent et j'ai souvent fait des recherches sur internet afin de comprendre comment tout cela fonctionne. EPITA m'a ainsi permis de mieux comprendre et d'enrichir mes connaissances dans ce domaine et de me permettre à mon tour de pouvoir développer des outils, des applications ou des jeux comme le projet de S2 et ainsi pouvoir passer d'observateur et utilisateur à développeur. Ce projet de S3 est une bonne opportunité pour emmagasiner d'avantages de connaissances qui seront utiles pour la suite de mes études et de réfléchir par moi-même et en groupe sur un projet plus sérieux qu'est cet OCR.

1.2.4 Baptiste *ZaitQA* Zammit

Login : *baptiste.zammit*

Un projet est à prendre au sérieux. C'est un devoir qui se prépare, qui se réfléchit et qui se travaille autant en solitaire qu'en groupe. Durant plusieurs années, j'ai eu la chance d'en mener deux avec un groupe de travail plus que convenable. Nous avons eu la possibilité de réfléchir, travailler, et évoluer ensemble au cours de la réalisation de nos projets pour à chaque fois donner un résultat qui nous correspondait. Je n'ai jamais eu la chance de réaliser un projet qui touchait essentiellement à l'informatique et pourtant, c'est ce que je désirais le plus avant cette année. J'ai rejoint EPITA dans l'optique de développer mes capacités, de pouvoir acquérir une bonne réflexion et trouver des solutions comme un ingénieur en travaillant en groupe pour avancer plus rapidement et d'une meilleure façon. Ce deuxième projet au sein d'EPITA me permettra a l'aide de mes coéquipiers, de solidifier et d'approfondir les notions acquises.

1.3. Présentation du projet

Le projet de cette année est un OCR. Il s'agit, en français, d'une reconnaissance de caractères qui est capable d'analyser une image contenant du texte afin de le détecter et de le fournir à l'utilisateur.
Une définition précise de ce qu'est un OCR serait, selon Futura Sciences :

Optical Character Recognition - Reconnaissance Optique de Caractères : technique qui, à partir d'un procédé optique, permet à un système informatique de lire et de stocker de façon automatique du texte dactylographié, imprimé ou manuscrit sans qu'on ait à retaper ce dernier.

Un logiciel OCR permet par exemple à partir d'un texte scanné, d'extraire la partie textuelle des images, et de l'éditer dans un logiciel de traitement de texte. Nous avons deux soutenances pour le réaliser entièrement et ce rapport explique le travail pour la première. Notre projet s'appelle *Socrate*. Nous avons choisi ce nom car il s'agit d'un jeu de mots avec les initiales "OCR" qu'il contient.

1.4. Outils utilisés

1.4.1 Programmation

Traitement de texte

- Vim
- Visual Studio Code 2017
- Atom

Compilation

- Gcc

Partage du projet

- Git
- GitHub

1.4.2 Environnement de développement

- Arch Linux
- Manjaro
- Ubuntu
- MacOS

1.5. Espérance d'avancement

1.5.1 Première soutenance

	Espérance
Segmentation de l'image	100%
Traitement de l'image	70%
XOR	100%
Réseau Neuronal	10%
Interface	20%
Apprentissage de réseau de neurones	20%

1.5.2 Deuxième soutenance

	Espérance
Segmentation de l'image	100%
Traitement de l'image	100%
Réseau Neuronal	100%
Interface	100%
Apprentissage de réseau de neurones	100%

Chapitre 2

Première soutenance

2.1. Objectifs et tâches à accomplir

Pour cette première soutenance, plusieurs missions et réalisations sont demandées.

Premièrement, la partie traitement et découpage de l'image doit être entièrement terminés. Cela comprend le chargement de l'image, sa transformation étape par étape jusqu'à une image possédant uniquement des pixels noirs et des pixels blancs. Enfin, sa découpe, d'abord en blocs de textes puis ensuite en lignes et enfin en caractères pour pouvoir les gérer séparément et les identifier. C'est une des deux grandes parties dont il faut s'occuper pour cette première soutenance.

La seconde est le réseau neuronal plus simple bien entendu, il doit être capable d'apprendre la fonction XOR nommée en français "OU EXCLUSIF". Pour cela, il faut l'initialiser, l'implémenter et enfin le faire apprendre. Au-delà de ces deux contraintes majeurs, le travail a avancé et à avoir commencé à la suite de ces deux tâches était la sauvegarde ainsi que le chargement de poids, trouver une bibliothèque d'images pour l'apprentissage et enfin la manipulation de fichiers pour la sauvegarde entière

des résultats. Vous l'aurez compris, cette première soutenance est chargé en défis et en objectifs, passons maintenant a l'explication détaillée.

2.2. Traitement de l'image

Partie réalisée par Merotto Enzo et Ajili Wassim.

2.2.1 Chargement de l'image

La première étape de notre OCR est de charger une image pour ensuite pouvoir l'utiliser en accédant à ses pixels.

Pour cela, nous avons décidé d'utiliser la bibliothèque disponible sur les ordinateurs d'EPITA : SDL. Cette bibliothèque permet de faciliter la manipulation d'images.

En effet, quelques lignes de commandes suffisent pour charger une image. SDL permet également de modifier facilement les images grâce à un accès à la totalité de leurs pixels ainsi que leurs composantes.

Par exemple, on peut accéder aux composantes RGB de chaque pixel pour connaître sa couleur et la modifier.

Les principales commandes de la bibliothèque SDL utilisés sont :

IMG_Load : *Charge une image en une surface exploitable par SDL*

SDL_GetRGB : *Récupére les composants RGB d'un pixel.*

2.2.2 Suppression des couleurs

La suppression des couleurs est la deuxième étape du traitement de l'image. Il est impératif d'enlever les couleurs de celle-ci afin qu'elle soit constituée uniquement de pixels noir et blanc pour faciliter la reconnaissance du texte.

Nous avons réalisé deux fonctions majeures pour réaliser ce travail, une fonction nommée "Grayscale" qui transforme l'image en niveau de gris. Le principe de cette fonction est simple : elle parcourt toute l'image et observe les composants RGB de chaque pixel. Pour transformer chacun de ces pixels en gris, chacun des composant doit être égale à la même valeur S. Cette valeur varie en fonction de la couleur initiale du pixel. En effet, elle se calcule comme cela :

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

Il y a ensuite une seconde fonction nommée "BinaryColor" qui à partir des différents niveaux de gris ne distingue plus que deux couleurs : le blanc et le noir. Cette fonction consiste juste à mettre les pixels avec les composants RGB inférieur à 128 en blanc et ceux supérieur ou égale à 128 en noir. Elle permet de simplifier l'étude de l'image.

Démonstration de nos fonction Grayscale et BinaryColor

2.2.3 Détection de blocs de textes

Cette partie fut l'une des plus longues à réaliser. La détection de blocs doit permettre de connaître l'emplacement des différents paragraphes de texte et des images pour ensuite les replacer après la retranscription. Ceci permet aussi de faciliter la détection des lignes qui compose chaque bloc de texte.

L'algorithme choisi est le RLSA qui signifie "Run Length Smoothing Algorithm". Cet algorithme est particulièrement efficace lorsqu'il s'agit de partitionner des documents ayant une mise en page de type Manhattan (textes/images alignés et séparés par des espaces blancs, comme par exemple les articles de journaux) mais s'avère moins fiable lorsqu'on est en présence de mises en pages complexes (image imbriquée dans du texte par exemple). Son principal défaut tient au fait qu'il utilise uniquement des informations locales lors du lissage ce qui peut entraîner l'apparition de liens entre le texte et les images.

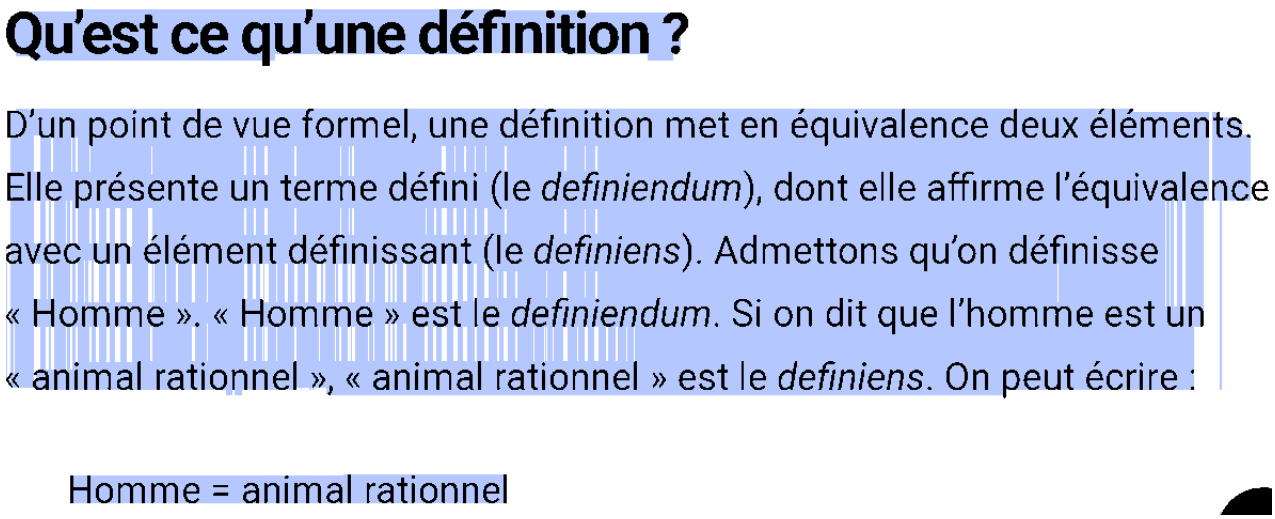
Le fonctionnement de cet algorithme est le suivant. On calcul premièrement un seuil C pour chaque ligne et chaque colonne de pixel qui sera utile pour la suite. Il se calcule ainsi :

$$C = \frac{nb\_de\_pixels\_blancs}{nb\_de\_pixels\_noirs}$$

Ensuite, nous commençons par effectuer un parcours horizontal sur une copie de l'image original. La fonction findBloc_H, prenant notre image en paramètres, permet de faire ce balayage. En effet, on parcourt l'image de gauche à droite dans l'objectif de trouver un pixel noir. Une fois que l'on a trouvé un pixel noir, on vérifie si le prochain pixel noir se trouve à une distance strictement inférieur au seuil C. Si oui, tous les pixels blanc entre les deux pixels noirs deviennent noir également, sinon on les laisse tels qu'ils

sont. Notre seconde étape est un balayage vertical, nous avons donc une fonction qui s'appelle findBloc_V. Celle-ci a les mêmes paramètres que findBloc_H et les mêmes principes, cependant, on parcourt la copie de notre image verticalement.

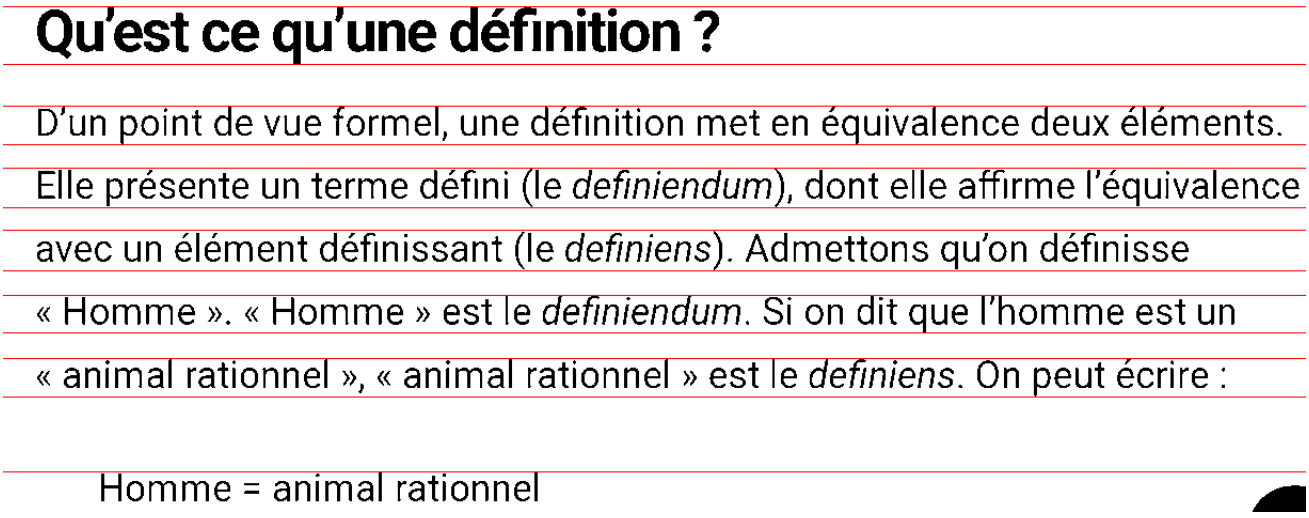
La fin de ces deux étapes se marquent par l'obtention de deux images : une avec des bandes noires horizontales remplaçant les textes (findBloc_H) et une avec des bandes noires verticales (findBloc_V). La troisième sera, naturellement, de mettre en commun les deux images. Nous allons donc utiliser la fonction VH. Cette fonction vérifiera si le pixel issu de l'image résultante du balayage horizontal est noir, tandis que l'on vérifiera si celui issu de l'image résultante du balayage vertical est noir également. Si les deux conditions sont vérifiés, le pixel résultant sur la copie de l'image original cette fois-ci devient noir (bleu clair dans notre cas), sinon il devient blanc. On fait cela pour tous les pixels des deux images, on enchaîne ensuite avec un second balayage horizontale, et ainsi, nous avons délimité les zones de textes.



Démonstration de notre traitement de blocs

2.2.4 Détection de lignes

Une fois que tous les blocs de texte sont détectés, il faut séparer toutes les lignes qui les composent. Pour cela, il est nécessaire de parcourir toutes les lignes de pixels et de distinguer les lignes qui contiennent des pixels noirs de celles qui n'en ont pas. Une ligne est donc représentée par un ensemble de lignes de pixels contenant des pixels noirs, encadrées par des lignes n'en contenant pas.



Démonstration de notre traitement de lignes

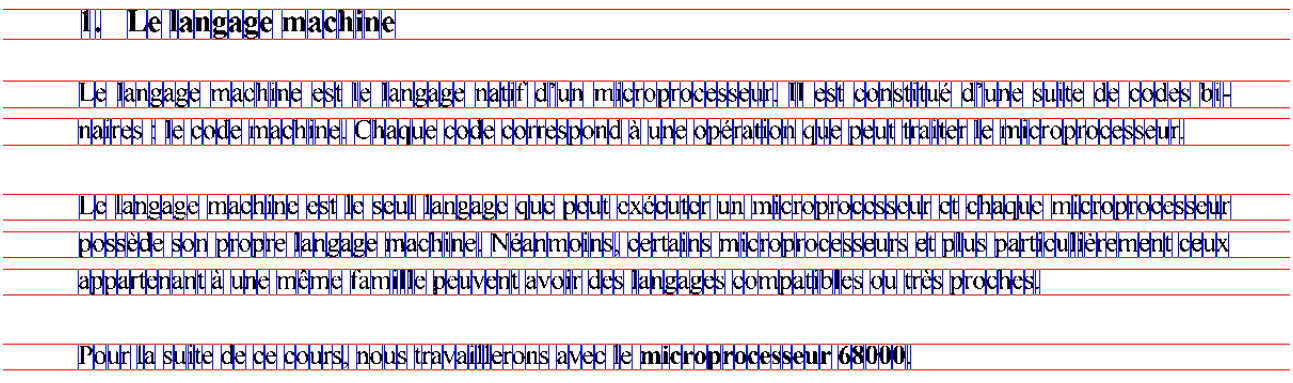
2.2.5 Détection de caractères

Il suffit enfin de parcourir toutes les lignes ainsi détectées afin de trouver tous les caractères.
Pour cela, on utilise la même méthode que pour les lignes, mais cette fois-ci, on observant les colonnes de pixels et non les lignes.

Après cela, le programme isole les caractères sur des images séparées. On obtient ainsi une multitude d'images dont chacune contient un caractère du texte. Une fonction va ensuite convertir ces images en tableau de 1 et de 0 (1 pour les pixels noirs et 0 pour les pixels blancs). C'est ce tableau qui sera envoyé au réseau neuronal afin de reconnaître le caractère en question.

Mais il n'y a pas que les caractères à reconnaître. En effet, il y a également les espaces et les retours à la ligne. Pour détecter les espaces, le programme parcourt chaque ligne de texte et calcul la moyenne de la taille de l'espace blanc entre deux caractères. Ainsi, si une zone blanche entre deux caractères a une taille supérieure à cette moyenne, il s'agit d'un espace.

Enfin, on détecte un retour à la ligne à chaque fois que l'on termine de parcourir une ligne de texte.



Démonstration de notre détection de caractères

2.3. Réseau Neuronal

Partie réalisée par Jérémie Zeitoun et Baptiste Zammit

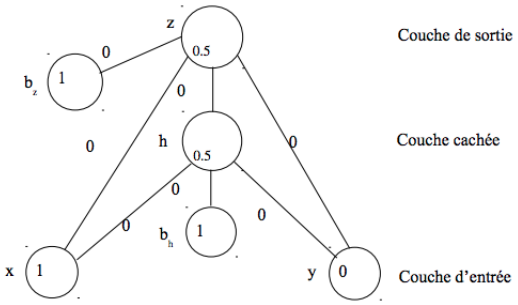
2.3.1 Initialisation du réseau neuronal

Le réseau neuronal que nous devions implémenter était un XOR plus communément appelé le "OU EXCLUSIF". Pour cela, nous avons choisi une structure comprenant six neurones. Les neurones créés suffisent à apprendre la fonction et à retourner des résultats corrects et encourageants pour la suite du projet. Pour cette première version du XOR, la forme utilisée est unique et non modifiable et permet de calculer uniquement le XOR

Ces six neurones sont :

- Deux entrées, x et y.
- Une sortie z.
- Trois neurones cachés, h, le biais de h et le biais de z.

De plus, nous avons une valeur attendue et un tableau de poids.



XOR avant l'apprentissage

2.3.2 Déroulement du XOR (apprentissage)

Ce XOR utilise l'algorithme "backprop", qui utilise la rétropropagation qui modifie les valeurs des poids par modification successives, il se déroule en plusieurs grandes étapes :

- Modification des entrées et de la valeur attendue.

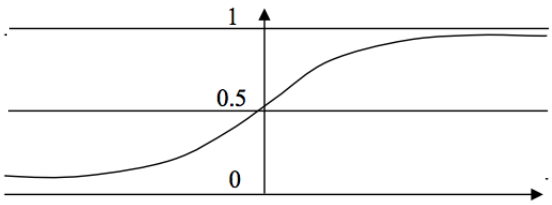
- Changement des valeurs d'activation :

La première étape est d'effectuer la somme du produit des poids de toutes les connexions entrantes ($w_{x,i}$) de chacun des neurones par leur valeur d'activation actuelle (o_i), on note cette somme net_x :

$$net_x = \sum_i w_{ix} o_i$$

Une fois cette somme effectué, on lui applique la fonction de transfert du réseau de neurones, dans le cas de ce réseau la fonction de transfert est la fonction sigmoïde de la forme :

$$f(x) = \frac{1}{1+e^{-x}}$$



Fonction sigmoïde

On obtient ainsi la nouvelle valeur du neurone d'activation qui servira pour la suite du XOR.

- Calcul de l'erreur de la valeur d'activation du neurone de sortie en fonction de la valeur attendu (w_z) :

$$d_z = (w_z - o_z)o_z(1 - o_z)$$

Puis modification de ses poids en fonction du pas (p) :

$$\Delta w_{iz} = p\ d_z\ o_i$$

De même, on calcule l'erreur de la valeur d'activation des neurones cachés :

$$d_h = o_h(1 - o_h)\sum_k d_k w_{kh}$$

Ici, on peut voir que l'erreur du neurone caché se calcule en fonction de sa valeur d'activation, des erreurs des neurones arrivants et des poids les reliant.
On modifie ensuite les poids comme pour le neurone de sortie.
À la fin de ces quelques étapes la valeur des poids se rapprocheront plus de la valeur qu'ils doivent atteindre pour donner les bonnes valeurs du XOR.
Une itération est une répétition de ces étapes pour chaque valeur d'entrée avec sa valeur attendue. n itérations seront donc n répétitions de l'algorithme avec les valeurs (0, 0, 1), (0, 1, 1), (1, 0, 1), (1, 1, 0), l'algorithme sera donc parcouru n * 4 fois.
Afin de tester le XOR il suffit d'effectuer les deux premières étapes et regarder la valeur d'activation du neurone de sortie. Sa valeur ne sera pas exactement 0 ou 1 mais une valeur s'en approchant grandement (ici à quelques centièmes).

x	0	0	1	1
y	0	1	0	1
z	0.5	0.5	0.5	0.5

Tableau des valeurs de sortie avant apprentissage

x	0	0	1	1
y	0	1	0	1
z	0.01	0.98	0.98	0.02

Tableau des valeurs de sortie après apprentissage

w_{bz}	w_{xz}	w_{xh}	w_{hz}	w_{bh}	w_{yh}	w_{yz}
-4.49	-7.87	8.25	16.44	-3.54	8.24	-7.87

Tableau des valeurs des poids après l'apprentissage

2.4. Travail à faire

2.4.1 Partie "Traitement de l'image"

Élément à améliorer

Pour la prochaine soutenance, plusieurs fonctions déjà implémentées devront être améliorées ou modifiées.

Tout d'abord il y a la fonction binaryColor. En effet, il faut la rendre un peu plus polyvalente pour gérer plus de cas. Par exemple si un texte clair est présent sur l'image, il sera difficile de le détecter avec l'implémentation actuelle de la fonction car il risque d'être effacé.

De plus, la détection de blocs est un autre point à améliorer. Il faut réussir a identifier les images pour n'analyser que le texte. Il faut également séparer les blocs de texte pour les retranscrire correctement.

Il faut également améliorer la détection de caractères afin de distinguer les caractères même lorsqu'il y a deux caractères collés.

Éléments à ajouter

Il y a également plusieurs fonctionnalités qu'il faudra ajouter pour la prochaine soutenance.

Premièrement, une fonction permettant de dimensionner les images peut s'avérer utile pour les afficher correctement en adaptant leur taille à notre future interface.

En effet, un autre ajout qu'il faudra faire est l'ajout d'une interface qui permettra de guider l'utilisateur afin de faciliter l'utilisation de notre

OCR.

Il peut être également intéressant de réaliser une fonction de redressement de l'image afin de remettre les lignes de texte de l'image droites. Cela est essentiel pour permettre à l'OCR de détecter et reconnaître le texte d'une image même s'il est tordu.

On pourra également ajouter des fonctions pour éliminer le bruit parasite des images ou encore renforcer les contrastes. Cela peut améliorer la netteté de l'image, et ainsi améliorer la reconnaissance du texte.

2.4.2 Partie "Réseau neuronal"

Dans un premier temps, le réseau de neurones va être passé en structure afin de pouvoir le modeler plus facilement et de créer d'autre réseaux de neurones que le XOR.

La sauvegarde et le chargement des poids sera effectué à chaque utilisation du réseau de neurones.
Le réseau de neurones sera complété afin de pouvoir reconnaître des caractère grâce à un apprentissage poussée avec une grande banque d'image de caractère dans des polices différentes, ou des images avec des caractères manuscrite.

En entrée le réseau de neurones recevra une table de 0 et de 1 correspondant au caractère à reconnaître et comparera chaque pixel au pixel de l'image type d'un caractère. Une variable va stocker la probabilité de correspondance de chaque caractère et sera stockée dans un tableau. Il suffira donc de regarder la plus grande probabilité pour choisir le bon caractère.

2.5. Conclusion

2.5.1 "Tableau d'avancement"

	Espérance
Segmentation de l'image	100%
Traitement de l'image	70%
XOR	100%
Réseau Neuronal	10%
Interface	20%
Apprentissage de réseau de neurones	20%

	Travail réalisé
Segmentation de l'image	90%
Traitement de l'image	50%
XOR	100%
Réseau Neuronal heightInterface	0% 0%
Apprentissage de réseau de neurones	0%

2.5.2 Partie "Traitement de l'image"

Notre plus gros problème fut la détection de bloc. Notre objectif était trop vaste, nous avons voulu traiter trop de cas particulier. Nous avons eu du mal à analyser nos recherches et bien les comprendre. Beaucoup de fonctions de test avec différentes méthodes ont été créées pour en fin de compte ne pas être retenus. Le reste du traitement d'image nous parait faisable et intéressant à faire.

2.5.3 Partie "Réseau neuronal"

Le problème majeur de cette partie est la structure neuronale. En effet, le réseau neuronal présent dans notre OCR est en fait très peu modulable et adaptative mise à part la fonction XOR. C'est-à-dire qu'aucunes autres fonctions ne pourra être FACILEMENT implémentable dans le réseau qui est loin d'être une structure dynamique.

2.5.4 Ressenti général

Tout d'abord à propos du groupe aucun problème apparent, un groupe dynamique, persévérant, curieux. Cependant, le sujet du projet est un sujet assez compliqué à étudier, peu de connaissances vraiment utiles pour travailler, beaucoup de recherches personnelles à effectuer.

Le ressenti général du groupe est plutôt positif avec une motivation assez développé pour finir cette OCR de façon efficace. Nous espérons que le travail effectué jusqu'à maintenant sera utilisable et correct pour le reste du projet.

Chapitre 3

Deuxième soutenance

3.1. Objectifs et tâches à accomplir

Les objectifs pour cette soutenances sont principalement tournés vers le réseau de neurones ainsi que l'interface. En ce qui concerne le réseau de neurones, le plus important est au départ de créé une structure de neurones capables de permettre le construction de n'importe quel réseau avant de s'attaquer à son apprentissage. Pour ce qui est de l'interface, tout doit être fait à l'aide de Gtk, rien n'a été encore codé.

3.2. Traitement et segmentation de l'image

Partie réalisée par Wassim Ajili et Enzo Merotto

Pour ce qui est du traitement de l'image, nous n'avons pas ajouté beaucoup de fonctionnalités car le travail effectué était déjà performant. Cependant pour cette deuxième soutenance, nous nous sommes occupé de lier et adapter la partie traitement d'image et segmentation au réseau de neurones. En effet, nous avons créé une fonction "resize" et une fonction "resize_char" qui permettent de dimensionner les caractères en matrices de largeur et de hau-

teur égale à 28. Cela est primordial pour permettre au réseau neuronal de détecter ces caractères. En effet, le réseau neuronal dois toujours posséder le mêmes nombre d'entrées qui correspondent au nombres de pixels sur le caractères et donc aux nombre d'éléments dans la matrice correspondante. Il y a donc 28 x 28 = 784 entrées.

3.3. Interface graphique

Partie réalisée par Baptiste Zammit

3.3.1 Introduction

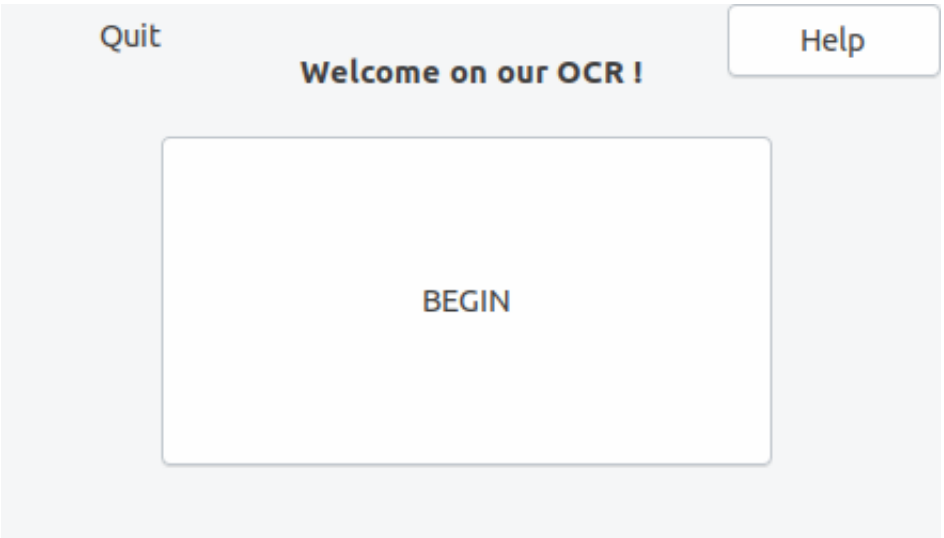
Pour cette deuxième soutenance nous devions créer une interface graphique pour pouvoir montrer le fonctionnement de notre OCR. Pour cela nous avons utilisé GTK 3.0 couplé à Glade, logiciel avec une interface graphique pour construire simplement et rapidement la nôtre. Le plus compliqué dans cette partie est que c'est la première fois que nous touchons à cette "technologie".

3.3.2 Glade

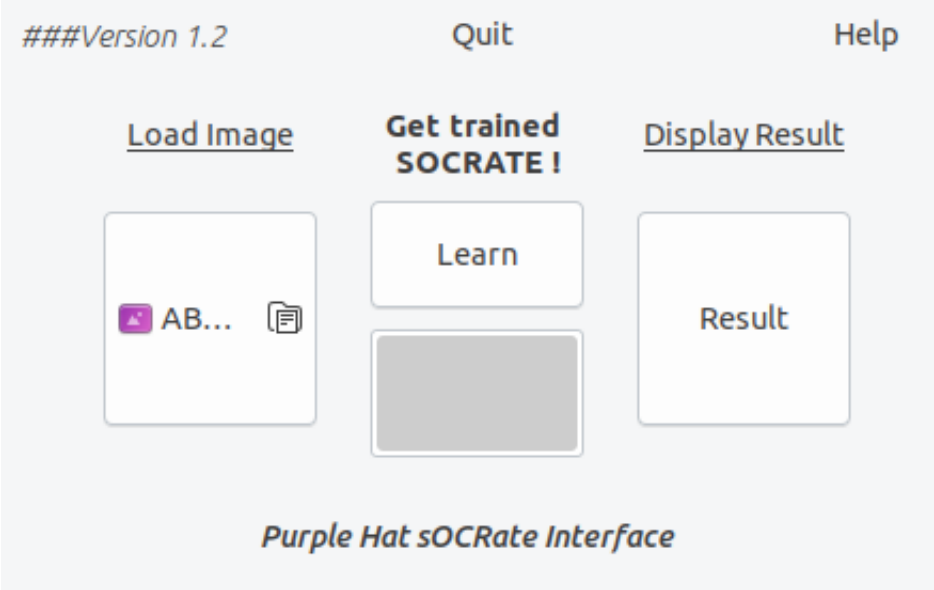
La première chose à faire c'est donc de construire notre environnement sur Glade pour ensuite l'atteler au code et avoir quelque chose de pleinement fonctionnel. Nous avons donc décidé d'avoir plusieurs fenêtres pour faciliter la compréhension aux utilisateurs et également donner un aspect attrayant au projet sur lequel nous travaillons depuis le début du semestre. Après de longues heures de travail nous avons une interface plutôt esthétique et assez fonctionnel.

3.3.3 GTK

Après avoir réalisé cette interface, il faut la connecter au reste du code pour pouvoir tout utiliser depuis celle-ci. Nous avons donc tout rassemblé dans notre fichier main ou nous pouvons appeler chaque fonction pour le bouton auquel elle est attribué. Une fois l'interface terminée, le travail s'achève sur de nombreux tests pour vérification du fonctionnement final.



Interface graphique



Interface graphique

3.4. Réseau Neuronal

Partie réalisée par Jérémie Zeitoun

3.4.1 Passage de cas particulier à structure

Pour la première soutenance le réseau de neurones que nous avions été sur uniquement sur un cas particulier qui reconnaît la fonction "XOR". En effet, modifier le nombre de neurones par exemple devait être fait manuellement afin de rajouter les poids 1 par 1 et ensuite les relier entre eux manuellement. Il était donc inconcevable de faire un OCR basé sur un réseau de neurone fait à la main sachant que l'on aurait besoin de plusieurs centaines de neurones d'entrées et dizaines de neurones cachés et de sortie, donc les relier entre eux un par un serait un travail beaucoup trop laborieux voire impossible. Il a donc été primordial de commencer par transformer notre réseau de neurones en structure qui sera beaucoup plus facile à manier, en modifiant une ou deux variable pour le restructurer complètement, ou rajouter des biais par exemple. Les fonctions de l'algorithme de propagation utilisent donc désormais des boucles pour parcourir tous les neurones du réseau et de manière automatique. La structure d'un neurones est composé de sa valeur, de sa couche, de la présence d'un biais relié au neurone et du poids correspondant. Il est aussi composé d'un tableau de poids entrant sur lui et sortant, ainsi qu'un tableau correspondant au neurones relié au poids entrant et au poids sortant ainsi que la valeur d'erreur pour les neurones de sortie. Plusieurs fonctions ont été rajoutés pour parcourir la structure, modifier tous les neurones d'entrées, récupéré les valeurs de sortie, les comparer à notre alphabet etc.

3.4.2 La base de données

La base de donnée que nous avons utilisé est composé de 62 caractères (minuscules, majuscules et chiffres). Grâce à un programme qui écrit dans un fichier les 0 et 1 correspondant à un caractère nous avons réalisé une bibliothèque de caractères qui sera utilisé pour l'entraînement du réseau de neurones.

3.4.3 Initialisation du réseau

La fonction permettant d'initialiser le réseau de neurones prend 3 paramètres correspondant au nombre de neurones pour la couche d'entrée, cachée et de sortie. Il crée donc autant de neurones que demandé et les relient entre eux en remplissant les tableaux de poids de chacun grâce à une fonction auxiliaire. Les différentes variables utilisées lors de l'entraînement du réseau sont aussi initialisées dans cette fonction.

3.4.4 Initialisation de la base de données

Nous avons utilisé une structure de caractères composé d'un tableau de flottant qui sera passé au réseau de neurones pour l'entraîner. Pour remplir ces tableaux de caractères nous avons eu recours à un "parser" que nous avons codé nous même afin de parcourir tous la base de données et de transposer les fichiers textes de 0 et 1 de chaque caractère en tableau de flottant pour ce caractère. L'ordre de la base de données est le même que celui de notre alphabet représenté par une string avec les 62 caractère utilisés.

3.4.5 Entraînement

Durant cette partie nous avons dût trouver les valeurs donnant les meilleurs résultat à la fin de l'apprentissage. Les valeurs à modifier qui influencent sur les résultats sont le nombre de neurones dans la couche cachée, le nombre d'itération de l'entraînement et le pas par lequel est multiplié chaque poids. Nous avons utilisé 3 boucles pour modifier ces 3 valeurs et relancé le réseau et l'entraînement pour garder les valeurs donnant le meilleur résultat. Après plusieurs heures à tourner nous avons obtenu les valeurs avec lesquels le réseau reconnaissait le plus de caractère et nous les avons gardés pour la suite. Enfin nous avons relié le réseau de neurones au reste du projet afin de pouvoir donner les tableaux de flottant de chaque caractère segmenté de l'image chargé au réseau entraîné et ainsi recevoir un résultat sur ce qu'il y a écrit dans l'image.

```
Avancement de l'apprentissage : 0 / 100
Avancement de l'apprentissage : 10 / 100
Avancement de l'apprentissage : 20 / 100
Avancement de l'apprentissage : 30 / 100
Avancement de l'apprentissage : 40 / 100
```

Avancement de l'entraînement

```
O -> True
P -> True
Q -> True
R -> True
S -> True
T -> True
U -> False (a)
V -> True
W -> False (g)
X -> False (F)
Y -> True
Z -> False (a)
0 -> False (a)
1 -> True
2 -> True
3 -> True
4 -> True
5 -> True
6 -> True
7 -> True
8 -> True
9 -> True
```

Test de reconnaissance de chaque caractère de la base de données

Pas	Itérations	Neurones cachés	Caractères reconnus
0.200	850	66	48
0.222	850	60	36
0.195	850	67	41

Résultat pour différentes valeurs

Chapitre 4

Conclusion Générale du projet

4.1. "Tableau d'avancement"

	Travail attendu
Segmentation de l'image	100%
Traitement de l'image	100%
Réseau Neuronal	100%
Interface	100%
Apprentissage de réseau de neurones	100%

	Travail réalisé
Segmentation de l'image	100%
Traitement de l'image	90%
Réseau Neuronal	100%
Interface	80%
Apprentissage de réseau de neurones	80%

4.2. Partie "Traitement et segmentation de l'image"

Le travail déjà effectué dans cette partie était déjà suffisant pour la première soutenance. En effet, le programme est capable de détecter toutes les lignes du texte et d'y découper tous les caractères séparément. Nous n'avions donc pas grand chose à ajouter pour cette soutenance. Nous avons donc pu nous concentrer sur le réseau neuronal qui, lui en revanche, a nécessité bien plus de temps. Cependant, si nous en avions eu plus, nous aurions pu ajouter quelques fonctionnalités bonus qui auraient amélioré la segmentation de l'image. Par exemple, une fonction permettant d'enlever le bruit de l'image ou encore une autre pour réorienter l'image afin que les lignes de textes soit toujours droites.

4.3. Partie "Réseau neuronal"

Le réseau neuronal a pris un temps assez conséquent a être mis en place mais les notions de bases du fonctionnement d'un réseau de neurones ont été acquises et pu être mise en pratique. Nous regrettons peut-être de ne pas assez avoir entraîné notre réseau et de ne pas lui avoir donné assez de police à apprendre. Mais dans l'ensemble, mis à part un résultat peu probant, la structure neuronal d'un OCR a bien été comprise et mise en place.

4.4. Partie "Interface graphique"

Pas de problème en particulier pour cette partie concernant l'interface. C'était la première fois que nous utilisions GTK et Glade, nous avons donc découvert leur utilisation. Finalement nous avons une interface qui nous plait et que nous pensons bien organisée.

4.5. Ressenti général

Cette période de travail qui s'est écoulée depuis la première soutenance a été une période difficile pour de multiples raisons. En effet, nous avons eu peu de temps pour réaliser un travail important et nous devions bien nous organiser pour réaliser ce projet sur notre temps libre en plus du travail qu'impose cette deuxième année de l'Epita. Cependant, nous avons donné tout notre possible pour obtenir un OCR le plus performant possible et même si le résultat n'est pas parfait, nous somme fier du travail accompli et nous avons beaucoup appris pendant ce projet. Que ce soit en traitement et manipulation d'images, en interface graphique ou en développement d'intelligence artificielle sous forme de réseau neuronal, nos connaissances se sont grandement accrues et nous ne finissons pas ce projet sans enseignements.

Chapitre 5

Sources

5.1. Pour le traitement de l'image

http ://crblpocr.blogspot.com/2007/06/run-length-smoothing-algorithm-sa.html

https ://pdfs.semanticscholar.org/c641/30cd6c0531ba8f5aa9759defeabf9b22ad16.pdf

https ://www.johndcook.com/blog/2009/08/24/algorithms-convert-color-grayscale/

https ://cs.stackexchange.com/questions/49447/how-to-binarize-an-image

5.2. Pour le réseau neuronal

http ://www.math-info.univ-paris5.fr/ bouzy/Doc/AA1/ReseauxDeNeurones1.pdf

http ://neuralnetworksanddeeplearning.com/

5.3. Pour l'interface graphique

https ://www.gtk.org/

https ://www.micahcarrick.com/gtk-glade3-gui-programming.html