Rapport de soutenance finale

OCR

by Feel The Code

Flavien Darche Alexis Piget-Domenger

Victor Dané Romain Schonholzer

Table des matières

1 Introduction

1.1 Un peu d'histoire ?

La première machine de reconnaissance optique de caractère fut créée par Gustav Tauschek, un ingénieur allemand, en 1929. Elle contenait un détecteur photo-sensible qui pointait une lumière sur un mot lorsqu'il correspondait à un gabarit contenu dans sa mémoire.

En 1950, Frank Rowlett, qui avait cassé le code diplomatique japonais PURPLE, demanda à David Shepard, (un prédécesseur de la NSA américaine), de travailler avec Louis Tordella pour faire à l'agence des propositions de procédures d'automatisation des données. La question incluait le problème de la conversion de messages imprimés en langage machine pour le traitement informatique. Shepard décida qu'il devait être possible de construire une machine pour le faire, et avec l'aide de Harvey Cook, un ami, il construisit « Gismo » dans son grenier pendant ses soirées et ses week-ends.

Ce fait fut rapporté dans le Washington Daily News du 27 avril 1951 et dans le New York Times du 26 décembre 1953. Shepard fonda alors Intelligent Machines Research Corporation(IMR), qui livra les premiers systèmes de ROC au monde exploités par des sociétés privées.

1.2 Le projet

Dans le cadre de notre spécialisation en informatique à l'EPITA, Quatres mois se sont écoulés depuis le commencement de ce projet. Le projet est un **OCR** *(Optical Character Recognition)*, ou **ROC** *(Reconnaissance optique de caractères)*.C'est une 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.

Ce rapport de soutenance décrit la nature du projet et le travail effectué durant ce semestre.

Ce projet est la création d'un logiciel de reconnaissance optique de caractères sous le langage C, et principalement dans l'environnement linux. Le logiciel doit être capable de détecter et d'extraire le texte d'une image source *(tel qu'une photographie, scan de document, etc)* et de restituer ce texte sous forme de document texte qui lui est utilisable contrairement au fichier image. Nous avons utilisé Git pour travailler simultanément sur le projet.

La conception d'un logiciel de ce calibre se fait en plusieurs étapes :

- 1) Pré-traitement de l'image : pour commencer, on s'occupe du pré-traitement de l'image qui nous permet d'avoir l'image idéale pour travailler dessus. Ceci peut inclure le redressement d'images inclinées et/ou déformées, des corrections de contraste, le passage en mode bicolore ou bien encore la détection de contours. Nous avons implémenté toutes ces méthodes à l'exception du cas où les images sont déformées.

- 2) Segmentation de l'image (ou Analyse de page) : Ensuite, vient le travail de segmentation. Cette étape permet d'isoler les éléments textuels, mots et caractères pour la reconnaissance.

- 3) Reconnaissance : Vient la partie la plus importante, la reconnaissance de caractère. Durant cette étape on se prononce sur l'identité d'un caractère à partir d'un apprentissage de sa forme.

- 4) Interface : Enfin comme tout logiciel, il faut une interface graphique ergonomique, permettant à un utilisateur quelconque de pouvoir utiliser intuitivement le logiciel.

2 Présentation de *Feel The Code*

2.1 Présentation du groupe

Nous sommes le groupe *Feel The Code* qui se compose de quatre membres : Flavien 'S0me1ne' Darche, Victor 'Kaowarstail' Dané, Alexis '66zombi99' Piget-Domenger, Romain 'Jaden' Schönholzer.
Nous nous sommes réunis pour faire ce projet qu'est l'OCR. Pour l'origine du nom de groupe, cela fait référence à notre symbiose incroyable entre le code et nous.

2.2 Flavien 'S0me1ne' Darche

Passionné depuis de nombreuses années par l'informatique, je développe en C# (ou dans d'autres langages) des applications sur mon temps libre. Le fait d'avoir été chef de projet dans le projet de S2 *(jeu vidéo)* me permet de savoir gérer une équipe et de mener à bien un projet. Le développement que j'effectue sur mon temps libre me permet d'avoir pu expérimenter de nombreuses choses en programmation sur le langage C qui est le langage principal de ce projet. Cette expérience permet de structurer le projet et de le commencer avec des bases solides sur lesquelles les membres de *Feel The Code* se basent.

2.3 Victor 'Kaowarstail' Dané

Enthousiasmé par les projets de groupe, c'est avec dévouement que je me lance dans cette nouvelle aventure. Avec ce projet j'espère consolider mes acquis en programmation et assimiler de nouvelles notions pour pouvoir progresser dans ce domaine vaste et complexe. Car comme le dirait le générique de Pokémon "I want to be the very best like no one ever was", citation me guidant à travers mes études à l'EPITA. Cependant, j'ai une fâcheuse tendance a me dissiper, mais bien canalisé, je travaille efficacement, c'est pour cela que j'aime bien les projets de groupe, car il m'apporte cette canalisation.

2.4 Alexis '66zombi99' Piget-Domenger

Je m'appelle Alexis Piget-Domenger, élève en classe A2 et membre de ce projet informatique. Je suis motivé pour porter jusqu'au bout ce projet. Étant un novice en programmation jusqu'à mon arrivée à l'Epita (spécialité SVT), j'ai pris un réel plaisir à effectuer mon projet de second semestre. Cela m'a permis d'apprendre de façon ludique la programmation par le biais de la création d'un jeu vidéo. Ce nouveau projet codé en langage C, que je ne connaissais pas au début de ce semestre, est moins ludique et plus sérieux que le précédent. Cependant il est plus représentatif et enrichissant que le projet de jeu vidéo. Il m'a permis d'apprendre plein de nouvelles choses à moi et à mes amis, en faisant un projet très enrichissant.

2.5 Romain 'Jaden' Schönholzer

Romain alias Jaden, je suis dans la classe de # depuis le S1#. Après le projet du S2 me voici embarqué dans une autre aventure avec un groupe différent. Faire un jeu vidéo était quelque chose de nouveau, tout comme l'est l'OCR. J'ai abordé ce projet avec beaucoup d'excitation et d'attente autant sur le plan humain, car j'étais persuadé que notre groupe allait fonctionner, que sur le plan technique. Sur ce dernier j'avais énormément d'inconnues et j'appréhendais de ne pas être à la hauteur. Comme le semestre dernier ce projet nous a apporté beaucoup. Je suis très heureux d'approcher un peu de ce qu'est l'IA et le *deep learning* qui était encore il y a quelque mois que des mots qui avait un sens très vague, des notions en soit très abstraites. Ces notions sont très importantes et peuvent potentiellement faire parties de mon avenir, la motivation était donc encore plus présente que pour les projets passés.

3 Traitement de l'image

3.1 Chargement de l'image

Aux prémices du projet, l'utilisateur pouvait charger son image via arguments via ligne de commande. Cependant nous avons choisi de réaliser une interface graphique. Le chargement se fait désormais sur celle-ci via un bouton spécifique. L'image est ensuite chargé en mémoire et grâce à la bibliothèque SDL (qui est une bibliothèque de traitement d'image). Tout les traitements de l'image sont effectués sur celle-ci.

3.2 Image en noir et blanc

Pour effectuer un découpage de caractères de qualité et effectuer une binarisation, il nous faut tout d'abord passer l'image de l'utilisateur en noir et blanc. Pour cette transformation nous avons utilisé un algorithme qui permet de transformer l'image en niveau de gris.

Cet algorithme va alors transformer toutes couleurs RGB en un niveau de gris variant du blanc au noir. Nous avons utilisé l'algorithme de Luminance qui à chaque pixel de l'image affecte un nouveau pixel dont la couleur est définie par la formule suivante :

$$\text{Luminance} = 0{,}2126 * \text{Rouge} + 0{,}7152 * \text{Vert} + 0{,}0722 * \text{Bleu}$$

(a) Avant (b) Après

FIGURE 1 – Texte avant et après l'application du niveau de gris

3.3 Binarisation de l'image

Afin d'obtenir une image composée uniquement de blanc et de noir, il faut effectuer une binarisation sur celle-ci. La binarisation d'une image est une opération qui renvoie une image constituée uniquement de deux classes de pixels *(ici le blanc et le noir)*.

Cependant il existe deux types de binarisation :

— La Binarisation par seuillage
— La Binarisation par segmentation

Nous avons choisis d'utiliser la binarisation par seuillage dans ce projet car celle-ci produit toujours deux classes de pixel. Mais dans cette binarisation par seuillage il existe deux méthodes de seuillage :

— Le seuil fixe : Le seuil qui permet de décider si un pixel est blanc ou noir est le même pour toute les images
— Le seuil automatique : L'analyse de l'image permet de déterminer un seuil adapté à celle-ci

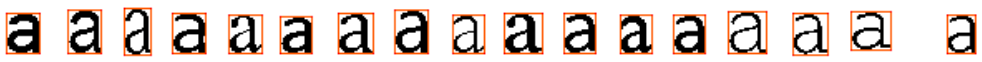
Dans ce projet pour obtenir de meilleurs résultat, nous avons opté pour le seuil automatique. L'algorithme que nous avons retenu qui nous permet d'implémenter cette méthode est celle d'*Otsu*. Pour déterminer le seuil de binarisation de l'image, cette méthode créer un histogramme par l'analyse des niveaux de gris de l'image. L'algorithme suppose alors que l'image à binariser ne contient que deux classes de pixels (le premier plan et l'arrière plan) et calcul ensuite le seuil optimal qui sépare ces deux plans afin que leur variance intra-classe soit minimale.

(a) Avant (b) Après

FIGURE 2 – Texte avant et après la binarisation de l'image

3.4 Segmentation de caractères

La segmentation de caractère est l'étape finale avant de faire fonctionner le réseau de neurones. Pour la première soutenance nous sommes partis sur des cas idéaux. Cela veux dire que sur chaque images, les caractères ne se touchent pas et chaque lignes sont séparer avec au minimum une ligne blanche de 1 pixel.



(a)

FIGURE 3 – Découpage de caractères

3.4.1 Détection des lignes

Pour détecter les lignes sur l'image de l'utilisateur, l'algorithme que nous utilisons parcourt verticalement l'image, ligne par ligne, à la recherche d'un caractère noir sur la ligne parcourue. Lorsqu'un pixel noir est détecté alors la ligne est marquée comme le haut d'une ligne de caractères. L'algorithme descend donc jusqu'à rencontrer une ligne totalement vide (remplis de pixels blanc) qui sera marquée comme la dernière ligne de la ligne de caractères. La ligne est donc découpée et ensuite cette ligne de caractères est passée dans l'algorithme de découpage de caractères.

3.4.2 Détection des caractères

Comme précédemment dans l'algorithme de détection de lignes, l'algorithme parcourt l'image mais ici il le parcourt horizontalement. Le principe est le même, lorsqu'une colonne de pixels contient un pixel noir alors c'est le début d'un caractère. La fin du caractère est définie lorsque une colonne entière de pixels blanc est trouvée ou lorsque la fin de l'image a été atteinte. Lorsque le caractère a été détectée, une matrice de pixels est créée et est placée dans une liste chaînée qui sera ensuite donnée au réseau de neurones pour la reconnaissance de celui-ci.

3.4.3 Précision

Le défaut des algorithmes de détection est la précision. En effet dans un mot, en fonction de la typographie, toute les lettres ne font pas la même taille *(par exemple, si la lettre est en majuscule ou en minuscule)* et ne sont pas forcément alignées *(par exemple les lettres g et t)*. Le découpage sera donc étendu vers le haut et le bas de la ligne et ne représentera pas exactement un caractère. Pour corriger ce problème une nouvelle exécution de l'algorithme est effectuée sur cette matrice de pixels (correspondant seulement au caractère). Cette nouvelle exécution permet d'avoir un découpage net et précis du caractère. La matrice de pixels du caractère est donc mise à jour.

3.4.4 Redimensionnement

Après la segmentation de caractère, chaque caractère est représenté par une matrice de pixels de taille différente. Cependant pour reconnaître un caractère le réseau de neurones a besoin d'une taille de matrice fixe.

Cette taille a été fixé par un carré de 28 pixels de coté.

Pour effectuer ce redimensionnement nous avons utilisé l'algorithme de l'interpolation au plus proche voisin. L'algorithme d'interpolation au plus proche voisin consiste à calculer une valeur approchée d'une fonction en un point quelconque à partir des valeurs de la fonction données en des points définis. L'algorithme est parfois déconseillé car celui-ci crée un fort effet de crénelage or la matrice de pixels envoyé au réseau de neurones n'est pas assez grande pour avoir un effet contraignant sur le résultat attendu.

3.4.5 Espaces et sauts de ligne

Le texte final retourné a l'utilisateur doit être le plus proche possible de ce qu'il voit sur l'image. Le texte final ne doit donc pas être une longue ligne illisible de caractères sans espaces.

Lors de la segmentation nous stockons dans un tableau le nombres de pixels qui séparent chaque caractères, ainsi à la fin de la segmentation nous pouvons effectuer une moyenne qui nous permettra de définir où l'espace doit avoir lieu. Les sauts de ligne sont également stockés. Lors de la restitution du texte à l'utilisateur les sauts de ligne et les espaces sont appliqués sur le texte.

3.5 Réduction du bruit

L'un des processus pré-traitement de l'image de notre OCR est la réduction du bruit. La réduction de bruit se fait après avoir binarisé l'image. L'algorithme que nous avons utilisé est celui consistant à faire une moyenne des couleurs des 8 pixels voisins pour éliminer les imperfections aléatoires de l'image. L'image apparait lissé de ses imperfections.

(a) Image de chat avec bruit

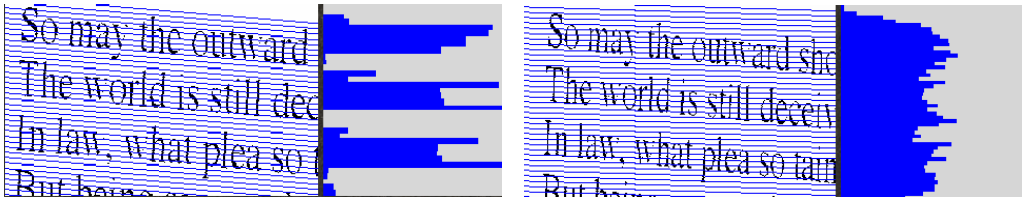
(b) Image de chat sans bruit

FIGURE 4 – Lissage de l'image

3.6 Redressement de l'image

Dans un monde idéal le texte sur l'image que nous souhaitons identifié est droit. Cependant le texte incliné existe aussi et nous nous devons de pouvoir le traiter. Il est donc nécessaire pour la segmentation de caractères de redresser l'image Pour trouver l'angle de redressement optimal, la méthode employée teste différents angles et projette la page afin de calculer un histogramme des pixels noirs.

Pour trouver un angle parallèle aux lignes du texte, il faut donc chercher à maximiser la variance de l'histogramme calculé à chaque projection. La première image [1] montre un cas où l'angle testé forme des lignes parallèles au texte, l'histogramme obtenu a donc une variance élevée. Tandis que la deuxième image utilise un angle résultant en des lignes obliques au texte, ce qui produit un histogramme avec une faible variance.



(a) Variance élevée (b) Faible variance

FIGURE 5 – Redressement de l'image

1. https://people.eecs.berkeley.edu/~fateman/kathey/skew.html

4 Interface utilisateur

Pour l'élaboration de ce projet, il nous paraissait indispensable de posséder une interface graphique le plus rapidement possible. En effet grâce à cette interface nous pouvons choisir l'image à analyser, voir visuellement les modifications apportées à l'image ou encore afficher le découpage de caractères.

Note : Pour cette deuxième soutenance l'interface utilisateur à été légèrement modifié pour inclure les différentes nouvelles fonctionnalités tel que la réduction de bruit.

Sur cette interface, vous pouvez y trouver dans la partie supérieur le nom du groupe, un bouton pour sélectionner une image dans l'ordinateur *(ouvre l'arborescence de fichier)* et une *image view* qui contient l'image que l'utilisateur a choisi. Dans le milieu de l'interface il y a 6 boutons.

1. Bouton "Greyscale" : permet d'effectuer la nuance de gris.
2. Bouton "Binarisation" : permet d'effectuer la binarisation.
3. Bouton "Deskew" : permet d'effectuer un ré-alignement de l'image si celle-ci est penché.
4. Bouton "Noise" : permet de réduire le bruit sur l'image et permettre une meilleur segmentation des caractères.
5. Bouton "Find chars" : permet d'effectuer le découpage de caractère. Les caractères découpés sont encadrés en rouge sur l'image.
6. Bouton "Recognize text" : permet de lancer la reconnaissance du texte sur le réseau de neurones.

Sur la partie inférieure de l'interface se trouve une grande zone de texte où sera affiché le texte reconnu par le réseau de neurones et deux boutons. Ces deux boutons permettront respectivement de sauvegarder le texte reconnu sous un fichier texte dans l'ordinateur et de remettre à zéro l'interface pour commencer une nouvelle reconnaissance de texte.

Les boutons centraux sont divisés en deux parties, la partie haute qui représente les étapes indispensable pour la reconnaissance des caractères sur l'image. La partie basse s'active lorsque l'étape de "binarisation" est terminé. Ces boutons permettent d'effectuer des pré-traitements d'images supplémentaires comme le ré-alignement de l'image si celle-ci est penché ou le fait de supprimé le bruit de l'image (supprimer des pixels indésirables qui pourrai gêner a la segmentation de caractères).

L'interface utilisateur a été crée à l'aide du logiciel Glade. Glade nous à permis de de concevoir cette interface visuellement et donc de pré-visualiser en temps réel à quoi le logiciel ressemblera. Glade produit un code en XML que nous donnons à la librairie GTK+3. Cette interface graphique a donc été réalisée avec Glade et la librairie GTK.

4.1 Expérience utilisateur

L'expérience utilisateur sur l'interface graphique a été pensée pour que les étapes de traitements de l'image soit le plus intuitif pour l'utilisateur. Chaque fois qu'une étape a été effectuée, le bouton qui a été cliqué devient grisé et n'est plus cliquable. Le caractère '' (checkmark) est ajouté sur le texte du bouton. Les boutons sont disposés dans l'ordre (de gauche à droite) en fonction des étapes. Les boutons des autres étapes ne sont pas disponible si l'étape actuelle n'a pas été effectuée. Le bouton *reset* remet à zéro l'état des boutons.
Notre but quand nous avons conçu cette interface graphique était qu'elle soit la plus simpliste et intuitive possible afin que n'importe qui puisse l'utiliser sans avoir besoin de compétences pré-requises

4.2 Sauvegarde du texte

Pour sauvegarder le texte sortie après la reconnaissance des caractère, un bouton a été prévu à cet effet. En effet ce bouton est positionné tous en bas de l'interface graphique. Le texte est ensuite sauvegardé dans un fichier texte au même endroit que la photo de base chargé par l'utilisateur.

Le fichier texte est sauvegardé sous le nom *"nom-image-ocr.txt"*.

5 Réseau de neurones

5.1 Introduction

L'utilisation d'un réseau de neurones est essentielle, voir capitale, pour l'OCR, en effet c'est à lui que revient la lourde tâche d'identifier les lettres afin de reconstruire le texte. Il s'agit d'une partie complexe à mettre en place car il agit comme un cerveau : il peut donc apprendre et se perfectionner et ainsi donner des résultats de plus en plus précis et proche de ceux attendus.

5.2 Qu'est ce qu'est un réseau de neurone ?

Un neurone formel est une représentation mathématique et informatique du neurone biologique. Il reproduit certaines caractéristiques biologiques, en particulier les dendrites, axone et synapses, au moyen de fonctions et de valeurs numériques.

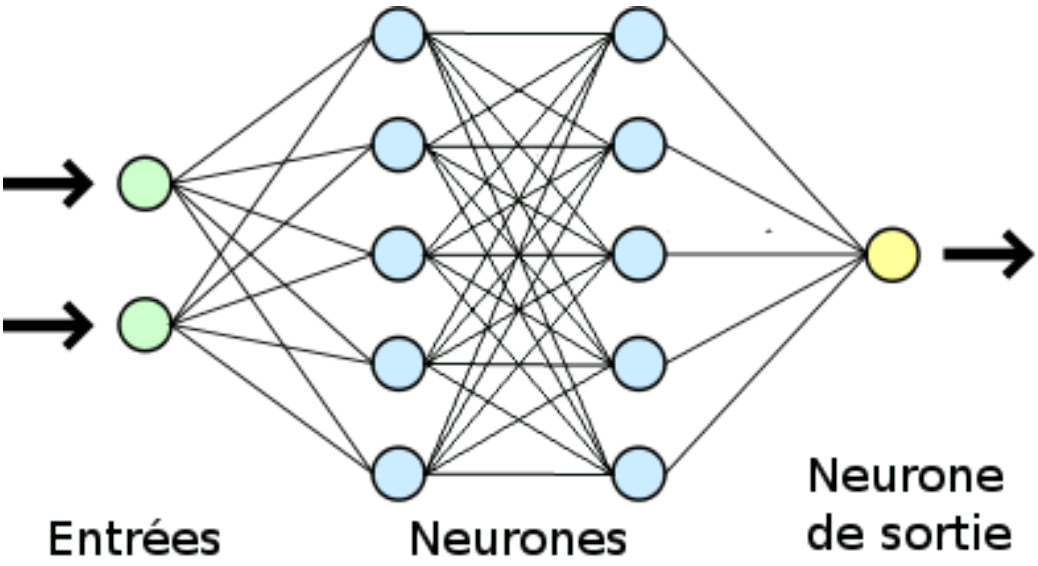


FIGURE 6 – réseaux de neurones

5.3 Principe de base

Un réseau de neurones se compose, comme son nom l'indique, de plusieurs neurones connectés entre eux. Chaque neurone à un ou plusieurs liens entrants et sortants. Ces liens, aussi appelés liaisons, sont affectés par des poids qui permettent de définir l'importance d'une liaison entre deux neurones par rapport à une autre. De plus, on rajoute à cela une fonction d'activation qui définie en fonction des variables d'entrées si un neurone est actif ou non. Le résultat de la fonction d'activation est renvoyé en sortie du neurone. On choisi généralement une fonction d'activation qui renvoie des valeurs entre [0 ; 1] ou [-1 ; 1].

En somme : un réseau de neurone peut être représenté par un ensemble d'entrées relié à un biais par des poids.
La fonction de combinaison retourne une valeur qu'elle a calculé en fonction des précédents paramètres qu'on lui a donné.
Ce résultat est passé en argument à la fonction d'activation dont la sortie sera la réponse du réseau de neurone.

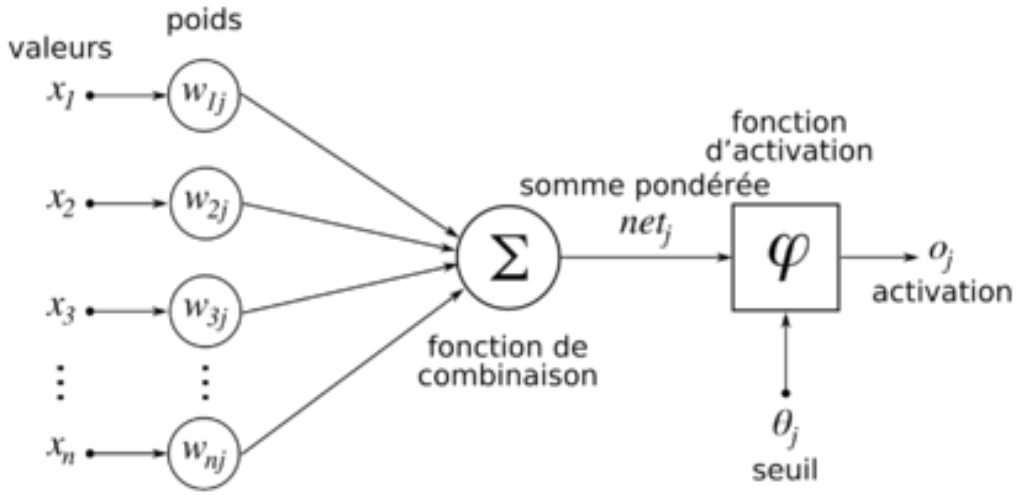


FIGURE 7 – neurone artificiel

Le biais est une fonction linéaire qui permet de savoir si le neurone doit être mis à 0 ou à 1 Pour la première soutenance, nous avions choisi la fonction *sigmoïde* comme fonction d'activation car c'est celle qui est la plus présente dans la documentation que nous avons trouvé. Pour la dernière soutenance, nous voulions nous pencher sur la fonction *softmax*.
Mais une fois sur le code nous nous sommes aperçu que nous avions assez de travail pour perdre du temps dessus sachant que l'impact sur notre projet aurait été moindre.

5.4 Entraînement

Pour que notre neurone apprenne les différentes lettres du texte nous avons du lui apprendre une *data base* grâce à une méthode se basant sur les pixels, c'est à dire que nous lui avons fait apprendre chaque position des pixels sur l'image, pour qu'il apprenne la forme de l'image.

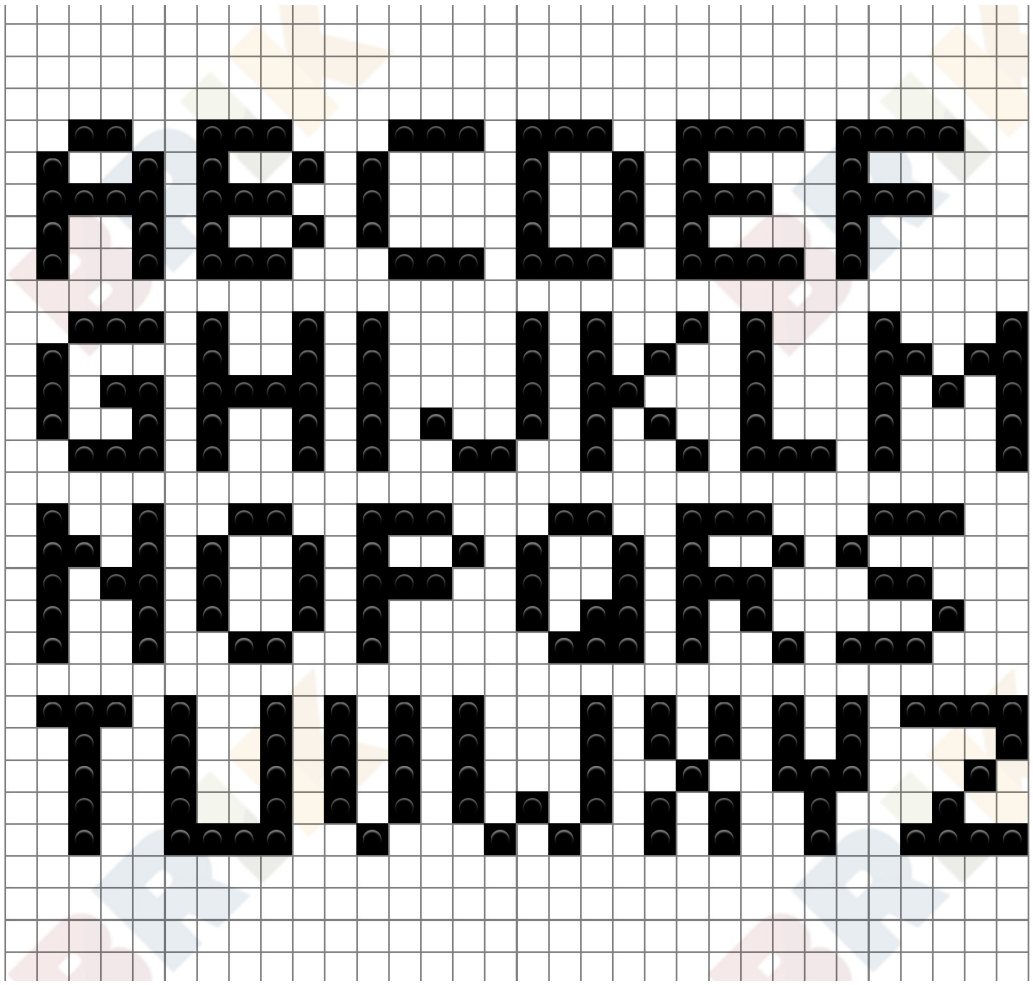


FIGURE 8 – alphabet en pixels

Nous avons ainsi récupéré la position de ces pixels noirs formant une matrice.
Étant donné qu'il existe différents style d'écriture pour une même lettre nous avons du créer un base de donnée regroupant 39 styles d'écriture différents pour à la fois les majuscules et les minuscules. Cela fait un total de 2028 images pour notre base de donnée !

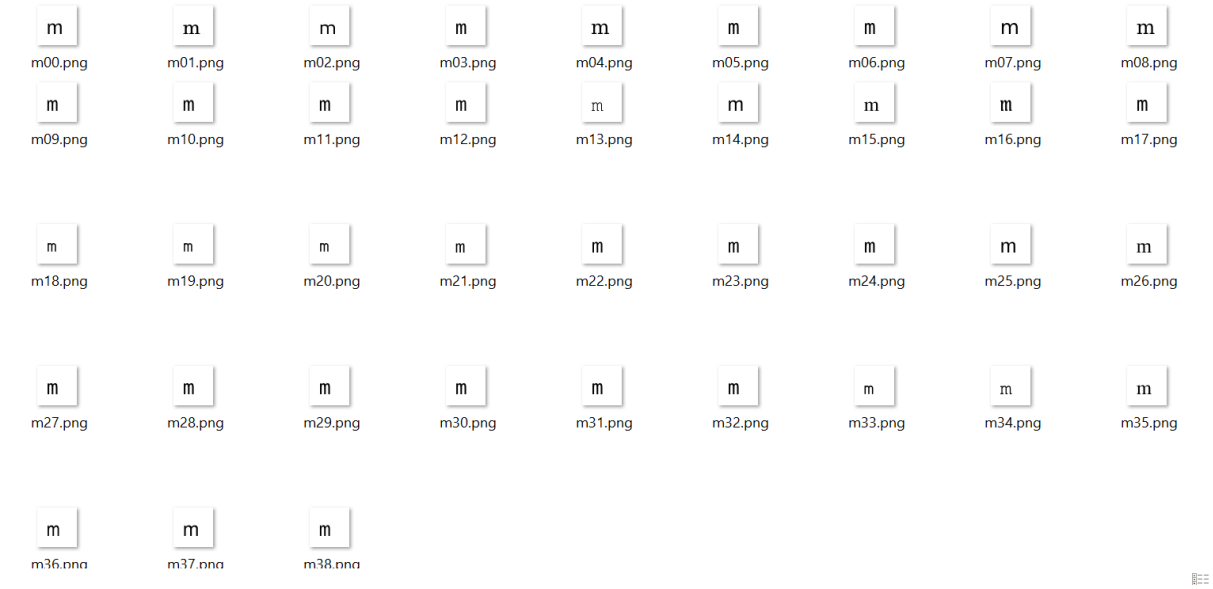


FIGURE 9 – base de donné du m minuscule

Pour appliqué cette matrice à chaque lettre nous avons dû fixés une résolution pour chaque image de lettre qui 28*28 ce qui donne un totale de 784 neurones en entrée.Notre réseaux de neurones à donc dû apprendre la position de ces 784 pixels pour nos 2028 images ce qui fait un totale de 1.589.952 position de pixels que notre réseaux de neurone à dû apprendre. Notre algorithme aura donc 52 neurones (26 lettres majuscules et 26 lettres minuscules) en sortie. Il reste cependant une couche caché appelé faisant de notre réseaux de neurones un *perceptrons multicouches*.Une couche caché n'a qu'une utilité intrinsèque pour le réseau de neurones et n'a pas de contact direct avec l'extérieur. Elle permet cependant de facilité et de perfectionner son traitement.

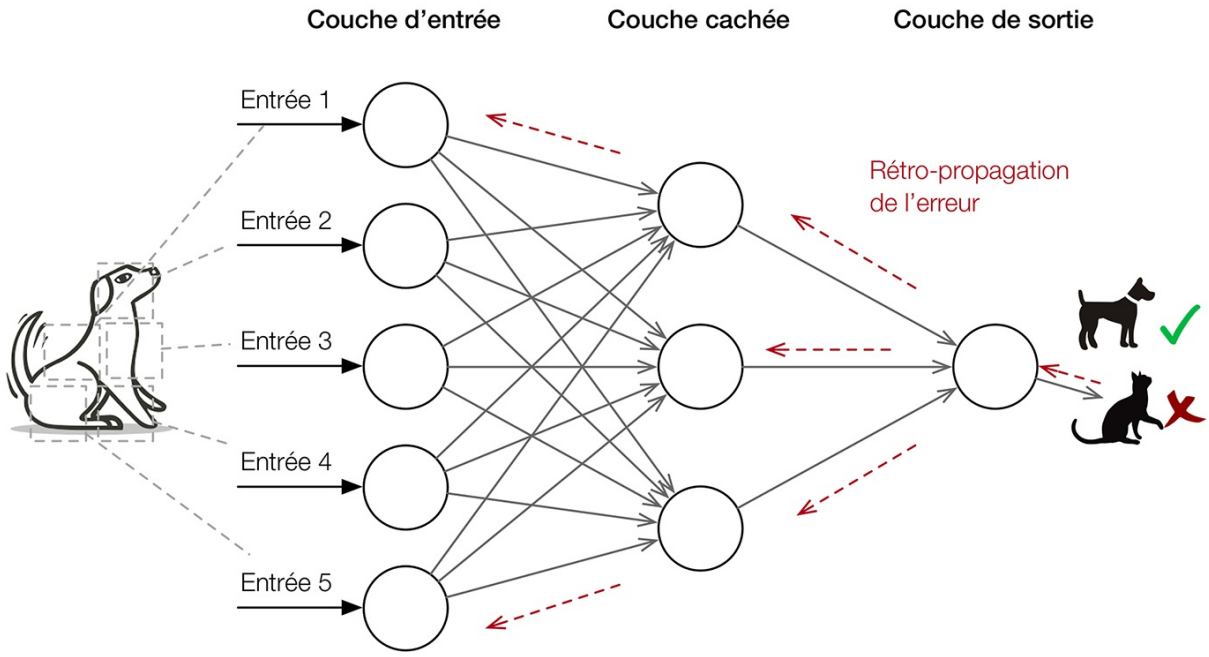


FIGURE 10 – différentes couches

Notre réseau de neurones possède lui 20 couches cachés.

L'entraînement de ce réseau de neurones lui à permit de s'améliorer : en augmentant la quantité des informations qu'on lui donne on affine la recherche pour avoir un résultat de plus en plus juste. Plus il y aura d'itération de l'algorithme plus les résultats seront justes.

Nous avons utilisé l'algorithme de rétro-propagation du gradient pour entraîner notre réseau, ce dernier se décompose en plusieurs étapes. On commence à lui donner un exemple à apprendre en entré. Ensuite on calcule les valeurs de sortie, puis on calcule l'erreur entre l'exemple et les valeurs d'activation des neurones. Enfin on met à jour les poids allant sur le neurone de sortie.

Plus cet algorithme sera répété, plus la sortie du réseau de neurone sera proche de la bonne solution.

Pour ce faire, nous avons injectée dans ce réseau des images, qui seront traités au préalablement par nos algorithmes, afin de faire marcher le réseau pour qu'il puisse s'améliorer.

5.5 Un exemple d'utilisation : la fonction XOR

Pour la première soutenance, nous avons utilisé notre implémentation du réseau de neurones pour lui faire apprendre la fonction XOR. Le réseau de neurones est composé de deux neurones d'entrées, d'un seul neurone sur la couche caché et d'un neurone de sortie.

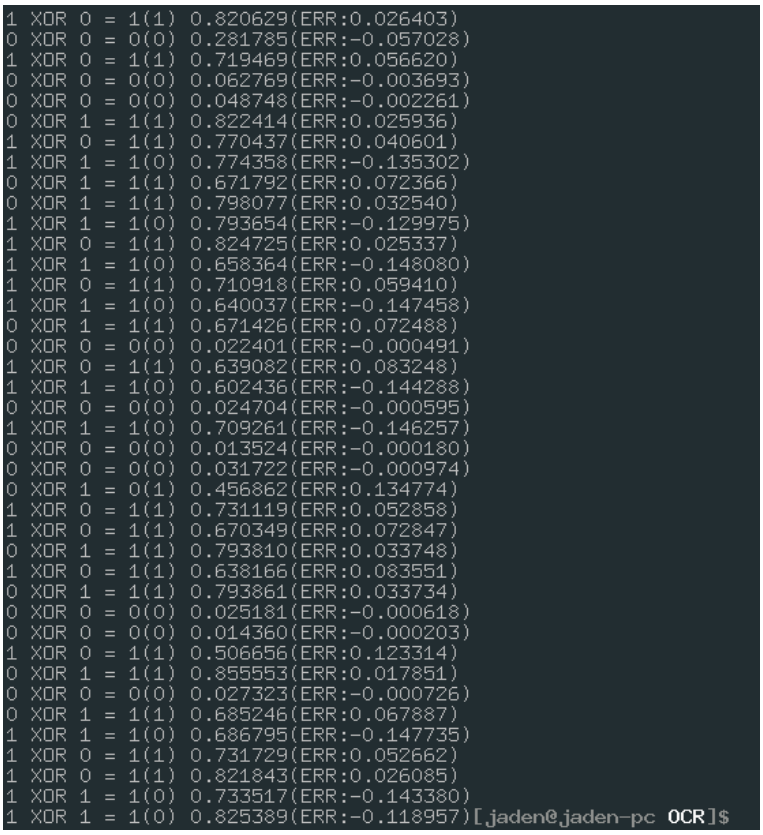


FIGURE 11 – taux d'erreur XOR

Comme on peut le voir sur la figure ci-dessus, le réseau de neurones apprend par rétropropagation à répondre correctement. Au bout d'un certain nombre de cycle d'apprentissage (appelé aussi epoch), le réseau de neurone est capable de reproduire la fonction XOR sans se tromper.

5.6 Création du jeu d'image pour l'entraînement du réseau de neurones

Plusieurs jeux d'images sont déjà disponibles sur Internet, mais la plupart sont utilisés pour des recherches avancées en traitement d'image et contiennent bien plus d'informations que nécessaires pour notre utilisation. Nous avons décidé de générer nos propres données ce qui nous donne un réel contrôle sur ces dernières.

Nous avons crée un script bash qui permet d'écrire une caractère voulu dans un fichier latex. Ce fichier latex est ensuite compilé. Le fichier pdf donné par le LaTeX est ensuite converti en image. Nous avons donc au final une image d'un caratère voulu. Dans ce script bash nous effectuons ceci pour toute les lettres majuscules et minuscules de l'alphabet pour toute les polices du système.

Toute la liste des polices du système est affiché lorsque l'ont tape dans un terminal *"fc-list : family"*.

Toutes les images sont sauvegardés dans un dossier nommé par le caractères en question et chaque image et nommé *char00.txt* où char équivaut au caractère et 00 à son numéro.

Cependant certaine polices d'écritures ne donne pas de résultats corrects pour certaines lettres de l'alphabet. Nous avons donc du regarder chaque images pour voir si elles correspondaient bien au caractère de base. Ces images ne possèdent aucun pré-traitement, ne sont pas à la bonne taille et aucun fichier texte avec la matrice de 1 et de 0 servant a l'apprentissage n'existe.

Au total nous avons un jeu d'images de 39 images par caractères.

Nous avons donc du implémenter une fonctionnalité dans l'OCR pour créer tous cela. L'argument "–createDataSet" de l'OCR permet de charger toutes ces images, d'effectuer les pré-traitements tel que la mise en noir et blanc, la binarisation et le redimensionnement et ensuite de créer le fichier texte.

(a) (b)

FIGURE 12 – Lettre avant et après traitement

```
jaden@jaden-pc OCR]$ ./main --createDataSet
raitment of: ImageSourceGenerator/fontsImages/A/A00.png
raitment of: ImageSourceGenerator/fontsImages/A/A01.png
raitment of: ImageSourceGenerator/fontsImages/A/A02.png
raitment of: ImageSourceGenerator/fontsImages/A/A03.png
raitment of: ImageSourceGenerator/fontsImages/A/A04.png
raitment of: ImageSourceGenerator/fontsImages/A/A05.png
raitment of: ImageSourceGenerator/fontsImages/A/A06.png
raitment of: ImageSourceGenerator/fontsImages/A/A07.png
raitment of: ImageSourceGenerator/fontsImages/A/A08.png
raitment of: ImageSourceGenerator/fontsImages/A/A09.png
raitment of: ImageSourceGenerator/fontsImages/A/A10.png
raitment of: ImageSourceGenerator/fontsImages/A/A11.png
raitment of: ImageSourceGenerator/fontsImages/A/A12.png
raitment of: ImageSourceGenerator/fontsImages/A/A13.png
raitment of: ImageSourceGenerator/fontsImages/A/A14.png
raitment of: ImageSourceGenerator/fontsImages/A/A15.png
raitment of: ImageSourceGenerator/fontsImages/A/A16.png
raitment of: ImageSourceGenerator/fontsImages/A/A17.png
raitment of: ImageSourceGenerator/fontsImages/A/A18.png
raitment of: ImageSourceGenerator/fontsImages/A/A19.png
raitment of: ImageSourceGenerator/fontsImages/A/A20.png
raitment of: ImageSourceGenerator/fontsImages/A/A21.png
raitment of: ImageSourceGenerator/fontsImages/A/A22.png
raitment of: ImageSourceGenerator/fontsImages/A/A23.png
raitment of: ImageSourceGenerator/fontsImages/A/A24.png
raitment of: ImageSourceGenerator/fontsImages/A/A25.png
```

FIGURE 13 – Traitement du jeu d'image par l'OCR

5.7 Apprentissage

Après la première soutenance nous avions une structure de base d'un réseau de neurones, obtenu grâce à notre fastidieuse élaboration du XOR. Bien qu'il était "basique" il nous a permis de ne pas repartir de 0. De ce fait nous nous sommes concentrés sur la mise en place de ce dernier.
Pour ce faire, nous avons mis en place un *Input layer* de 784 neurones pour prendre en charge la matrice de la lettre injectée dans le réseau. Ensuite nous avons un *Hidden layer* de vingt neurones qui après plusieurs boucles d'entraînement nous permet de retourner le résultat de notre réseau de neurone pris en charge par le *Output layer* de 52 neurones qui correspondent aux 2 fois 26 lettres de l'alphabet (les majuscules et les minuscules) pour pouvoir avoir un résultat en adéquation avec nos attentes.

Pour faire un entraînement efficace et exhaustif, nous utilisons les lettres précédemment générées. Pour ce faire nous appelons notre réseau pour chaque matrice de lettre, et cela trente neuf fois par caractère et cela cinquante deux fois pour couvrir l'ensemble des sortie de l'*Output layer* de notre réseau de neurones.

Voici un extrait d'un entraînement de notre réseau :

```
Position Found = 51 Expected 51 OK
Char entered: z | Char recoginized: z | ErrorRate: 0.000042
Position Found = 51 Expected 51 OK
Char entered: z | Char recoginized: z | ErrorRate: 0.000034
Position Found = 51 Expected 51 OK
Char entered: z | Char recoginized: z | ErrorRate: 0.000034
Position Found = 51 Expected 51 OK
Char entered: z | Char recoginized: z | ErrorRate: 0.000034
Position Found = 51 Expected 51 OK
Char entered: z | Char recoginized: z | ErrorRate: 0.000034
Position Found = 51 Expected 51 OK
Char entered: z | Char recoginized: z | ErrorRate: 0.000034
Position Found = 51 Expected 51 OK
Char entered: z | Char recoginized: z | ErrorRate: 0.000034
Position Found = 51 Expected 51 OK
Char entered: z | Char recoginized: z | ErrorRate: 0.000161
Position Found = 51 Expected 51 OK
Char entered: z | Char recoginized: z | ErrorRate: 0.000042
Position Found = 51 Expected 51 OK
Char entered: z | Char recoginized: z | ErrorRate: 0.000042
Position Found = 51 Expected 51 OK
Char entered: z | Char recoginized: z | ErrorRate: 0.000042
Position Found = 51 Expected 51 OK
Char entered: z | Char recoginized: z | ErrorRate: 0.000034
Position Found = 51 Expected 51 OK
Char entered: z | Char recoginized: z | ErrorRate: 0.000038
Epoch 1300  | MaxErrorRate =  0.967203
```

6 Avancée entre les deux soutenances

Soutenance	**1**	**2**
chargement des images et suppression des couleurs	fait	fait
détection et découpage en blocs, lignes et caractère	fait	fait
fonction XOR	fait	fait
sauvegarde et chargement des poids	en cours	fait
jeu d'images pour l'apprentissage	fait	fait
manipulation des fichiers (sauvegarde)	fait	fait
réseaux de neurones	en cours	fait
reconstruction du texte et sauvegarde	non	fait
une interface utilisateur	fait	fait
pré-traitement	en cours	fait
intégration d'un correcteur orthographique	non	non
bonus	en cours	partiellement

7 Problèmes rencontrés et pistes d'amélioration

Durant le développement de l'OCR nous avons rencontrés de nombreux problèmes. Cependant nous avons une idée de comment corriger certaine de ces erreurs et d'améliorer notre projet.

7.1 Problèmes rencontrés

Pendant notre projet, lorsque nous avons découvert l'utilisation des pointeurs, ils nous aient arrivé plusieurs fois de nous tromper sur l'implémention de la méthode "row-major order" où nous avons inversé la largeur et la hauteur (respectivement width et height).

Nous avons rencontré un problème lors du découpage des caractères. Nous avions pas pris en compte les points de certaine typo sur les i et les j ce qui nous a donné des formes très ressemblante que le réseau de neurone ne pouvait pas différencier même en l'entraînant longtemps. Comme nous avons douté d'abord de notre réseau de neurone nous n'avons pas vérifié le résultat de la segmentation qui était en fait déficient. Par exemple dans le dossier des images de j minuscule, le point sur le j et la barre du j sont coupé en deux caractères. Or seul le premier caractère (le point) est gardé lors de la création du jeu d'image, donc l'entraînement est biaisé par ces erreurs. C'est donc pour cela que nous possédons un taux d'erreur très élevé lors de l'entraînement.

Nous avons eu aussi des problèmes de motivation et de management que nous avons pu résoudre en n'ayant été sévère entre nous et en n'étant un minimum rigoureux.

7.2 Pistes d'amélioration et Solutions

Pour améliorer notre OCR, il faut fondamentalement refaire notre jeu d'image. Il faudrai prendre différente police d'écriture manuellement où nous avons vérifié que toute les lettres sont bien représentés et non pas d'utiliser toute les polices du système où certaine police remplace certain caractère de l'alphabet latin par d'autre caractères spéciaux. Il faudrai également modifier notre fonction de segmentation pour que le caractère en lui même ne soit pas coupé en deux.

Une piste d'amélioration supplémentaire est d'intégrer la reconnaissances des caractères spéciaux tel que les parenthèses ou les crochets ou de pouvoir reconnaître d'autre caractères comme par exemple d'autre alphabet.

8 Expériences personnelles

8.1 Flavien :

Durant ce projet, je me suis concentré sur tout ce qui était de l'ordre du pré-traitement de l'image, création de la structure du projet et la création de l'interface graphique.

J'ai également lié toute les parties de code des autres membres (tout le réseau de neurones avec l'interface graphique et les pré-traitements de l'image). Grâce à cette première partie de projet j'ai pu m'améliorer drastiquement dans le langage C. En effet grâce à la manipulation de librairie externe comme GTK ou SDL, j'ai pu lire beaucoup de documentation les concernant et apprendre a les utiliser, lier les codes du réseau de neurones m'a également permis de comprendre comment un réseau de neurones fonctionne.

Le fait d'être chef de ce groupe m'a permis de tisser des liens plus fort avec les autres membres pour une meilleure collaboration.

8.2 Victor :

Durant ce projet, je me suis principalement concentré, lors de cette seconde et dernière partie du projet, sur le réseau de neurones. Je m'occupais, en particulier, de l'implantation de ce dernier. Ce fut, personnellement, un véritable parcours du combattant car cela nécessitais de solides connaissances en C afin d'arriver à un résultat convenable. Évidement, ce ne fut pas le cas sur mes premiers tests, mais après de longues heures, que dis-je jours, de réflexion et de débug, je suis finalement arrivé à un résultat convenable. Ce projet m'a donc apporté une expérience très enrichissante, d'un point de vue "technique" avec toutes les notions de C utilisé, mais si ce ne fut pas une partie de plaisir, mais aussi d'un point de humain, car ce magnifique projet m'a permit de me rapprocher de mes camarades avec qui j'ai partagés cette formidable aventure humaine qui nous a permis de mieux se découvrir, se comprendre, s'apprécier et travailler ensemble.

8.3 Romain :

Pour conclure le projet de ce semestre ce fut bien plus instructif et intéressant que le projet du semestre dernier. J'ai eu l'impression d'être dans un océan de donnée et d'information que je ne comprenais pas, il fallait avoir une excellente vision d'ensemble pour savoir quoi faire et avancer.
Je sais qu'avec ce réseau de neurone je n'ai fait qu'effleurer les possibilités qu'offrent l'apprentissage automatique et c'est assez excitant ! J'espère avoir la possibilité de m'y confronter de nouveau car le milieu m'intéresse encore plus
Pour la première soutenance, j'avais épluché beaucoup de documentation afin de vulgariser toutes les formules mathématiques relatif au réseau de neurones, qui paraissaient trop abstraites à mes camarades.
Pour cette dernière, il a bien fallu être plus pratique et mettre en application ce que j'avais lu. Merci à mon groupe d'avoir été présent et soudé jusqu'au bout de ce projet et avec qui j'ai appris encore une fois à travailler en groupe. Nous avons fini sans trop d'accroc et en collaborant. C'était une expérience humaine et technique vraiment enrichissante.

8.4 Alexis :

Je me suis particulièrement concentré sur le réseau de neurones durant cette seconde session de travail. Ce fut un travail très pénible du fait de devoir comprendre en quoi consistait le réseau de neurones. De nombreuses heures ont dut être sacrifiée en documentation pour maîtriser parfaitement ce sujet. Ce ne fut qu'avec l'aide de tout le monde que le réseaux, dernière partie essentielle de ce travail, fut finis.

Ce travail bien que fatiguant fut très enrichissant car il me permit de comprendre le fonctionnement d'un réseau de neurones. Ce projet fut très enrichissant à la fois sur le point de vue technique et sociale. Au niveau technique car il me permis d'apprendre plein de nouvelles choses pas forcements vu dans nos cours traditionnels mais aussi des compétence technique n'ayant rien à voir avec la programmation en elle même mais néanmoins indispensable au bon fonctionnement du projet. Au niveau sociale également, car on a beau faire des projets pour améliorer nos compétences de travaux en groupe depuis notre plus jeune âge, personne ne peut affirmer maîtriser parfaitement cette compétences et du points de vu de notre groupe je pense que nous nous sommes bien améliorer depuis le début du projet.

Cela fait de ce projet une réussite à mes yeux non pas seulement car notre projet marche mais car il nous à apporté à moi et à mes coéquipiers beaucoup de choses.Je suis très satisfais à la fois de mon groupe, de mon travail et de mon projet.Je tiens cependant à remercier Flavien ayant les meilleurs compétences de programmation de notre groupe car malgré tout nos désir de faire un bon projet il nous ait souvent arrivé à moi et au reste du groupe d'être coincés sur bout de programme que Flavien à toujours su résoudre et nous expliquer ensuite les erreur pour que nous progressions.

9 Conclusion

Ce nouveau projet nous a permis d'apprendre de nombreuses nouvelles compétences et d'enrichir notre savoir-faire en matière de programmation. En jetant un regard rétrospectif sur notre projet nous pouvons affirmer que nous avons gagner beaucoup d'expérience et que ce projet nous permettra de mieux appréhender notre première année d'ingénieur à venir.

Concernant le projet lui-même nous sommes plutôt satisfait du résultat obtenu, même si nous aurions évidemment voulu être en moyen de proposer un logiciel plus avancé, notre niveau de programmation n'étant pas encore suffisant pour faire mieux. Nous avons toujours su respecter en temps et en heure nos engagements vis-à-vis du logiciel à créer et vous fournir un travail approfondi. Lors de cette soutenance nous avons pu vous proposer une base solide concernant à la fois le traitement de l'image, la détection de texte et la reconnaissance de caractères.

Du fait que les lauréats du prix **Turing** (prix attribué à ceux ayant la plus importante contribution de nature technique faite à la communauté informatique) de cette année ont basé leur travail sur le *deep learning* et sur les algorithmes d'apprentissages, nous sommes encore plus convaincus de la pertinence de ce projet. Notre projet, à nous, est centré sur un réseau de neurones capable de comprendre des caractères. Les lauréats, eux, ont un réseaux de neurones capable de reconnaître des voix. Cette légère proximité nous a motivé à mieux comprendre de quoi ce projet retournait.

10 Bibliographie

Vous pouvez retrouver ci dessous les documents qui nous ont permis d'apprendre sur le réseau de neurones et de réaliser les pré-traitements.

10.1 Pré-traitements

— Réalisation du Grayscale :
https ://en.wikipedia.org/wiki/Grayscale

— Réalisation de la méthode d'Otsu pour la binarisation :
https ://en.wikipedia.org/wiki/Otsu%27s_method

— Algorithme de réalignement :
http ://leptonica.org/skew-measurement.html
https ://people.eecs.berkeley.edu/ fateman/kathey/skew.html

— Réduction du bruit :
https ://en.wikipedia.org/wiki/Noise_reduction

10.2 Réseau de neurones

— http ://neuralnetworksanddeeplearning.com/chap3.html
— https ://youtu.be/aircAruvnKk
— http ://helios.mi.parisdescartes.fr/ bouzy/Doc/AA1/ReseauxDeNeurones1.pdf
— http ://mediamining.univ-lyon2.fr/velcin/public/deep/perceptron_multi.pdf
— http ://neuralnetworksanddeeplearning.com/chap1.html
— https ://fr.wikipedia.org/wiki/R%C3%A9tropropagation_du_gradient
— https ://www.youtube.com/watch ?time_continue=274v=Ijqkc7OLenI