Rapport 1: OCR Sudoku Solver

Yann MESSE Jules HAQUIN Vanina BARAHONA UNDURRAGA

October 2022

Table des matières

1 Introduction

1.1 Présentation du groupe

1.1.1 Formation du groupe

Notre groupe est formé par 3 etudiants d'EPITA Strasbourg :
— Yann Messé
— Jules Haquin
— Vanina Barahona Undurraga

La création de notre groupe n'a pas été sans accrocs. En effet, notre groupe ne s'est pas formé de manière spontanée. La création du groupe a commencé lorsque Yann a demandé sur Discord s'il y avait des groupes incomplets ou si des personnes étaient encore toutes seules. Vanina a répondu à Yann, cependant les autres personnes sans groupe n'ont pas répondu. Jules nous a rejoint par la suite et nous avons donc eu un groupe de 3.

Cependant, dans le cahier des charges, il était spécifié que les groupes devaient être exactement de 4. Nous avons donc cherché une autre personne et demandé qui était déjà dans un groupe pour savoir, par élimination, qui n'était pas encore dans un groupe. Toutefois, personne n'a répondu à ce message et nous avons appris que nous étions 37 et non 40.

Nous avons donc demandé à Jimmy (la personne responsable de l'OCR à Strasbourg) comment faire. Nous avons donc reçu l'autorisation de créer un groupe de 3.

Ainsi notre groupe a vu le jour.

1.1.2 Ses membres

Yann :

Je suis étudiant à EPITA, en 2^e année du cycle préparatoire. J'ai fait un bac spécialité Mathématiques et NSI.

Durant ma première année, j'ai fait en projet de groupe un jeu 2D RPG dans un monde futuriste. Ce premier projet m'a permis d'acquérir l'expérience basique de projet en groupe, ce qui me permet de mieux me débrouiller pour ce projet.

En effet, ce nouveau projet que nous avons durant notre premier semestre de deuxième année de prépa est l'OCR. C'est un projet de traitement d'image. Un domaine dont je ne connais pas grand-chose et que nous allons devoir découvrir pour le mener à bien. Je suis persuadé que ce projet se déroulera dans une bonne ambiance.

Vanina :

Je suis actuellement étudiante en deuxième année du cycle préparatoire à l'EPITA. J'ai fait un bac spécialité Mathématiques et Sciences de l'Ingénieur.
L'année dernière, j'ai fait en projet de groupe un jeu de hack'n slash sur la mythologie grec intitulé khaos.
Ce nouveau projet me permet de progresser en C avec des notions importantes que j'ai encore du mal à maîtriser comme les pointeurs ; autour d'une thématique que je ne connais pas très bien : le traitement d'image.

Jules :

Actuellement étudiant en SPE à EPITA, j'ai eu un baccalauréat spécialité Mathématiques et Physique-chimie.
Je démarre ce projet de S3 avec de l'enthousiasme. En effet je trouve la thématique du traitement d'image et en particulier des réseaux de neurones intéressante. C'est un sujet qui me stimule intellectuellement, beaucoup plus que le projet de S2.
De plus, ce projet va me permettre de progresser en C, langage que je connaissais assez peu avant de rentrer en SPE.

1.2 Presentation du projet OCR

Ce projet entre dans le programme de la deuxième année de l'EPITA (École pour l'informatique et les techniques avancées) et est à réaliser en groupe de maximum quatre personnes. Sa durée est d'un semestre, de Septembre à Décembre. Ce rapport de soutenance décrit la nature du projet ainsi que les différentes parties qui ont, pour le moment, composé sa création.

La reconnaissance optique de caractère (de son abréviation anglaise OCR pour Optical Character Recognition), est un système qui, comme l'indique son nom, a pour but de détecter un texte à partir d'une image standard (soit non éditable) pour pouvoir le sauvegarder et ainsi le modifier.

Ce projet est composé de deux grandes parties : les éventuels traitements d'image ainsi que d'autres opérations sur ces dernières, puis la reconnaissance des caractères par un réseau de neurones. L'objectif du projet est de pouvoir coder un algorithme en langage C qui aura à la fin une utilisation facilitée grâce à une interface graphique.

1.3 Repartition des taches

1.3.1 Soutenance 1

Tâches	Yann	Vanina	Jules
Chargement d'une image et suppression des couleurs	*		
Rotation manuelle de l'image	*		
Détection de la grille et de la position des cases		*	
Découpage de l'image (sauvegarde de chaque case sous la forme d'une image)		*	
Implémentation de l'algorithme de résolution d'un sudoku (Solver)			*
Réseaux de neurones			*
A commencer			
Sauvegarde et chargement des poids du réseau de neurones			*
Jeu d'images pour l'apprentissage ;			*
Manipulation de fichiers pour la sauvegarde des résultats			*

Légende : '*' indique la personne qui s'occupe de la tâche.

1.3.2 Soutenance final

Tâches	Yann	Vanina	Jules
Le prétraitement complet	*		
Rotation manuelle de l'image	*		
Le réseau de neurones complet et fonctionnel			*
Apprentissage			*
Reconnaissance des chiffres de la grille			*
La reconstruction de la grille		*	
La résolution de la grille			*
L'affichage de la grille ;		*	
La sauvegarde du résultat			*
Une interface graphique permettant d'utiliser tous ces éléments	*		

Légende : '*' indique la personne qui s'occupe de la tache.

2 Taches réalisées

2.1 Prétraitement

Tout d'abord, pour cette partie, j'ai choisi d'utiliser la librairie SDL Image, déjà disponible sur les ordinateurs de l'école, car elle permet de simplifier grandement le traitement et la modification d'une image. Pour effectuer le pré-traitement dans la situation actuelle du projet, il faut aller dans le dossier prétraitement et taper la commande suivante :

```
$ make
$ ./pretraitement image.jpeg
```

Pour que le fichier soit sans les exécutables et les .o, vous pouvez faire

```
$ make clean
```

Chargement de l'image
Pour le chargement de l'image, j'ai utilisé la fonction load image de la librairie SDL image. Cette fonction crée une surface à partir d'un lien vers une image qui permet de charger des images .jpeg et .png. Le type surface est facile à utiliser, car il permet d'avoir accès à la largeur et hauteur de l'image, le type de format de l'image (le nombre de couleurs différentes), le type de pixel (RGB, RGBA,...),... qui sont des informations que nous utilisons souvent durant le pré-traitement.

Augmentation des constrastes
La première fonction que j'appelle lors de mon pré-traitement est la fonction pixel to gamma. Cette fonction augmente le contraste de l'image (le gamma). Pour cela, j'utilise la fonction de contraste qui est : $G = 255 * (couleur/255)^{gamma}$. Pour chacune des couleurs RGB soit les composantes rouges, vertes et bleues. Cette augmentation de contraste permet de faire que le fond soit moins confondu avec l'avant de l'image.

Nuance de gris
Après l'appel de la fonction qui augmente le contraste, nous passons l'image en nuance de gris. Pour cela, nous utilisons la formule de normalisation visuelle de couleur : $Gray = 0.299 * r + 0.587 * g + 0.114 * b$. Après, il suffit de mettre cette valeur aux 3 composantes RGB et le pixel est en gris, suffit de le faire pour tous les pixels de l'image.

Rotation de l'image

L'étape qui suit est la rotation. En effet, pour l'instant, la rotation est manuelle, ce qui veut dire que la rotation est effectué si et seulement si, lorsqu'on appelle l'exécutable pré-traitement, on ajoute un angle en ° après le lien pour l'image.

```
$ make
$ ./pretraitement image.jpeg 35
```

La rotation est un élément clé du pré-traitement et a été assez difficilement réalisée. En effet, pour effectuer la rotation, j'ai décidé de prendre le problème à l'envers. Pour cela, j'ai choisi de prendre une surface différente de celle que je veux copier et je regarde dans la surface originale la valeur du pixel dans la nouvelle surface. C'est pour cela que j'effectue, s'il y a rotation, les 2 premières étapes sur la première surface et pour le reste sur la 2ème qui sera celle que nous continuerons d'utiliser plus tard. Pour trouver la valeur du pixel dans la surface originale, j'utilise la trigonométrie.

Je trouve d'abord le centre de l'image, j'utilise les propriétés de la trigonométrie des cos et sin. Ainsi l'angle de rotation est la conversion de l'angle en ° en Rad. La conversion est la suivante $angle = theta * PI/180$.

Pour trouver les coordonnées dans la surface originale, on a les calculs suivants :

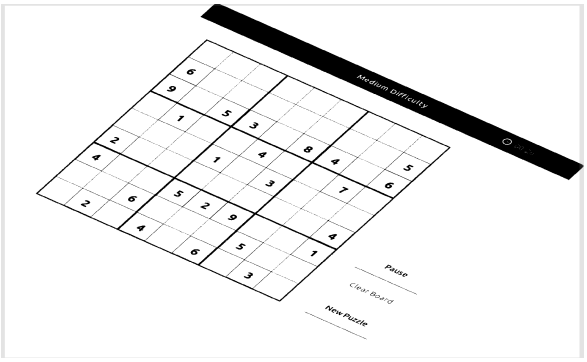
$x = cos(angle)*(i-verticalCenter)+sin(angle)*(j-horizontalCenter)+verticalCenter$

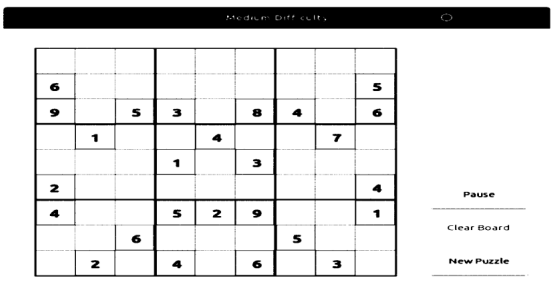
$y = -sin(angle)*(i-verticalCenter)+cos(angle)*(j-horizontalCenter)+verticalCenter$

La "nouvelle surface" est la "surface originale" avec la rotation de theta°, ainsi, on recherche dans la surface originale avec une rotation de -theta°.

Dans les formules, on a : i est la coordonnée verticale du pixel dans la "nouvelle surface". j est la coordonnée horizontale du pixel dans la "nouvelle surface". Nous avons aussi horizontalCenter qui est le centre horizontal de la surface et verticalCenter le centre vertical de la surface.

Cependant, la fonction ne s'arrête pas là, car les valeurs x et y ne sont pas entières. Donc, on récupère les valeurs entières qui entourent x et y et on applique une fonction mathématique qui est l'interpolation bilinéaire. Cette fonction permet d'obtenir la valeur du pixel en fonction des valeurs des pixels qui lui correspondent plus ou moins.





Filtre Gaussien

L'étape après la rotation est une étape qui permet de supprimer les pixels isolés, et qui en même temps rendent l'image un peu floue. Pour cela, je mets que la nouvelle valeur des pixels est égale à la moyenne des pixels autour avec un coefficient multiplicatif qui varie en fonction de la distance au pixel centrale. Pour ma part, j'utilise une matrice de 3*3 ce qui donne comme coefficient : $Gx = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} * A$

Binarisation

Pour la binarisation, il existe beaucoup de solution. En effet cette partie a pour objectif de trouver le seuil pour passer les pixels qui étaient en nuance de gris en noir et blanc.

Il existe beaucoup de technique et aucune d'elles ne fonctionne pour tous les cas.

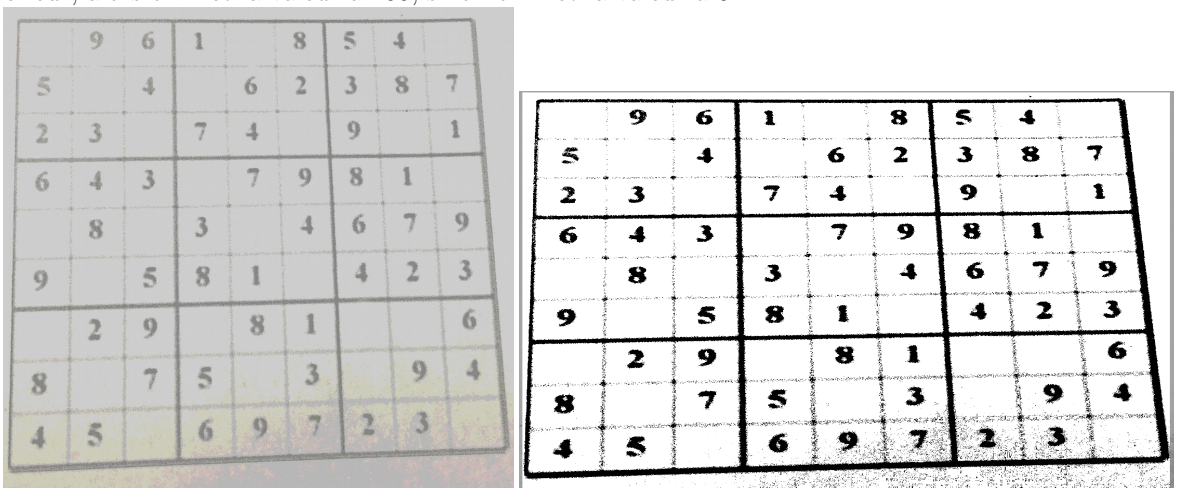
Pour ma part, j'ai décidé d'utiliser la méthode de Sauvola, qui permet de calculer des seuils locaux. C'est une amélioration de la méthode d'Otsu qui utilise le même principe, mais moins complet.

Pour ma part, j'ai décidé d'utiliser la méthode de Sauvola. Cependant, il existait un problème : $Seuil = M * (1 - (k * (1 - E/V)))$ Où nous avons M la moyenne locale, k une variable, E l'écart-type local et V la variance définie à 125 dans la méthode de Sauvola. J'ai pour ma part décidé de prendre un entourage de 11*11 pour le calcul du seuil local. Le problème rencontré étant, quel doit être la valeur de k ? Pour résoudre ce problème, j'ai décidé de prendre :

- En général k=0.4
- Si l'image est sombre alors k=0.36
- Si l'image est claire alors k=0.84
- Si le contraste de l'image est bas alors k = k - différence de luminosité de l'image sur la luminosité maximal de l'image +0.08
- Si le contraste de l'image est élevé alors k = k - différence de luminosité de l'image sur la luminosité maximal de l'image +0.6

Pour calculer la valeur de seuil, j'utilise une liste temporaire qui est initialisé pour qu'elle soit égale à l'image, initialise pour que la modification des valeurs de l'image ne modifie pas les valeurs de seuil. En effet, je calcule le seuil à partir de cette liste et non celle de la surface.

Après avoir calculé le seuil, on l'applique au pixel, et on regarde si la valeur du pixel est supérieur, alors on met la valeur à 255, sinon on met la valeur à 0.



2.2 Détection de la grille et découpage de l'image

2.2.1 Choix de la méthode

J'ai commencé par chercher une méthode de détection de bord. J'ai longtemps hésité entre la transformation de Hough et le Flood Fill algorithm. J'ai commencé par la transformation de Hough mais je me perdais. L'avantage du Flood Fill algorithm c'est qu'il utilise des notions déjà apprises en cours comme le parcours en profondeur ; donc plus facile pour moi a prendre en main. Cependant je trouvais cette approche moins pertinente, donc je me suis accrochée pour faire la transformation de Hough, que je n'ai malheureusement pas encore terminé.

2.2.2 Le filtre de Sobel

Pour détecter les lignes, il faut d'abord détecter les bords et produire une image de bord. J'ai pour cela utilisé l'algorithme de Sobel. Le principe est de déterminer où il y a des changements radicaux d'intensité dans l'image.

Tout d'abord on cherche a calculer le gradient G de chaque pixel.

Pour cela on calcule Gx le gradient horizontal et Gy le gradient vertical qui sont la convolution d'une matrice définie (kernel) et de A, une matrice 3x3 contenant le pixel dont l'on veut calculer le gradient et ses 8 voisins qui l'entourent.

Ici on manipule des matrices, mais le tableau de pixels étant en une dimension, on peut le voir en tableau de deux dimensions grâce au row major order.

$$Gx = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * A$$

$$Gy = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * A$$

Faire la convolution de deux matrices c'est multiplier point par point les éléments de même index des deux matrices, et d'en faire la somme. Voici ci-dessous un exemple quelconque :

$$\text{Posons } A = \begin{bmatrix} -3 & 4 & -2 \\ 1 & 6 & 0 \\ 0 & -2 & -3 \end{bmatrix}$$

$$\text{Alors } Gx = 1\times(-3)+0\times4+(-1)\times(-2)+2\times1+0\times6+(-2)\times0+1\times0+0\times(-2)+(-1)\times-3 = 4$$

Le problème est qu'un pixel ne contient pas une seule valeur, mais trois : r, g et b. Je fais donc la moyenne des trois pour en avoir une.

Une fois qu'on a Gx et Gy, on peut à présent calculer G.

$$G = \sqrt{(Gx)^2 + (Gy)^2}$$

Enfin on compare ce gradient à un seuil. Si G est en dessous du seuil alors c'est un bord et on met le pixel à 0 (noir) sinon ce n'est pas un bord et on met le pixel à 255 (blanc).

2.2.3 Transformation de Hough

La transformation de Hough est une technique de reconnaissance de formes qui nous permettra de détecter les lignes.

La fonction houghTransform prend en paramètre l'image de bord et un tableau d'entier en deux dimensions appelé accumulator. Cet accumulateur représente **l'espace de Hough**.

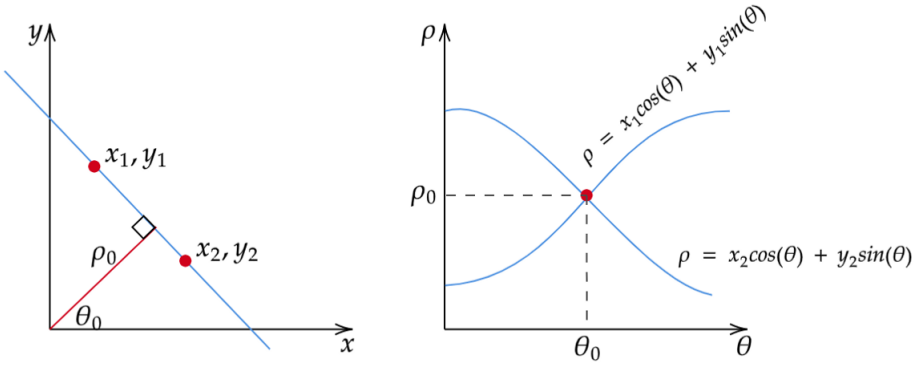


FIGURE 1 – Représentation d'une droite dans l'espace de Hough

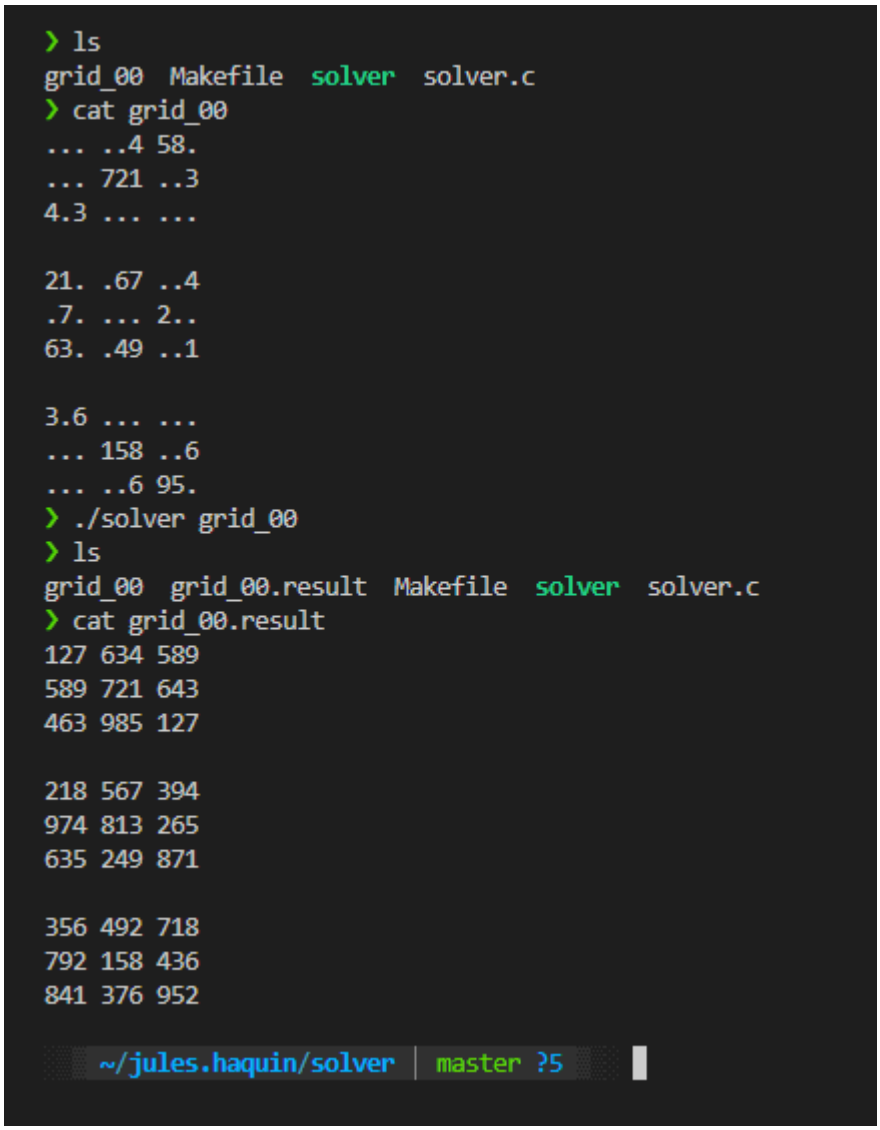
Une droite va être représentée par une ligne normale perpendiculaire à cette droite et qui passe par l'origine. rho (ρ) est la longueur de cette ligne normal et se calcule :
$\rho = x\cos(\theta) + y\sin(\theta)$
Thêta est son angle.
Là où deux points se coupent dans l'espace de Hough, il y a une droite.
Découpage de l'image
N'ayant pas encore fini la détection de la grille, je ne peux pas encore faire le découpage de l'image. Voici l'approche que je souhaiterais faire.

2.3 Solver

Pour la résolution de la grille de sudoku, nous avons choisi d'implémenter l'algorithme de backtracking. C'est un algorithme récursif que nous connaissons déjà bien, car nous avons fait un TP dessus l'année dernière en C#.
L'algorithme fonctionne de la manière suivante :

- on trouve une case vide
- on essaye de la remplir par chaque chiffre (1-9)
- on lance la récursion sur tous les chiffres qui font que la grille reste valide
- s'il ne reste plus de cases vides, la grille est résolue
- s'il n'y a plus d'appels récursifs alors qu'il reste des cases vides, alors la grille n'est pas solvable

Notre programme solver fonctionne de la manière suivante :


❯ ls
grid_00 Makefile solver solver.c
❯ cat grid_00
... ..4 58.
... 721 ..3
4.3 ... ...
21. .67 ..4
.7. ... 2..
63. .49 ..1
3.6 ... ...
... 158 ..6
... ..6 95.
❯ ./solver grid_00
❯ ls
grid_00 grid_00.result Makefile solver solver.c
❯ cat grid_00.result
127 634 589
589 721 643
463 985 127
218 567 394
974 813 265
635 249 871
356 492 718
792 158 436
841 376 952
~/jules.haquin/solver | master ?5

2.4 Réseaux de neurones

Nos deux réseaux de neurones ont une structure générale assez similaire, malgré le fait qu'ils aient deux buts totalement différents. Ils sont tous les deux composés d'une couche pour l'entrée, d'une couche cachée et d'une couche pour la sortie. La seule différence majeure réside dans le nombre de neurones pour chaque couche.
Ils peuvent se décomposer en deux grandes parties :

Forward-propagation :
Cette fonction consiste à calculer la sortie de chaque neurone de la couche cachée et de la dernière couche.
Pour cela, nous utilisons la fonction sigmoïde :

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

Sauf pour calculer la sortie de la dernière couche du réseau OCR, pour laquelle nous utilisons la fonction softmax :

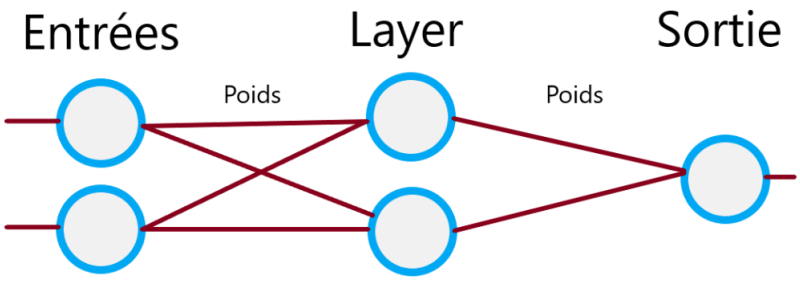
$$g(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$$

Cette fonction permet à notre réseau de prédire le nombre qu'on lui donne avec une probabilité entre 0 et 1.

Back-propagation C'est avec cette fonction que nos réseaux sont capables d'apprendre à résoudre un problème donné. Cette fonction est appelée après la forward-propagation. Elle permet d'ajuster les poids et les biais du réseau, en tenant compte de l'erreur effective qui est constaté après la forward-propagation.
L'entraînement des réseaux consiste donc à appeler la forward-propagation puis la back-propagation un certain nombre de fois (*epoch*), afin que les poids et les biais permettent à notre réseau de donner la bonne réponse.

2.4.1 XOR

Pour le XOR nous avons choisi d'implémenter un réseau avec deux neurones pour l'entrée, deux pour la couche cachée et un neurone pour la sortie. Nous pouvons représenter notre réseau par ce schéma :



Les résultats que nous obtenons avec 10 000 *epochs* et un taux d'apprentissage de 3 sont les suivants :

```
❯ ./xorNN
Network training with 10000 epochs and learning rate of 3.000000
XOR(0.000000, 0.000000) --> output = 0.005975, expected 0.000000
XOR(0.000000, 1.000000) --> output = 0.993875, expected 1.000000
XOR(1.000000, 0.000000) --> output = 0.993721, expected 1.000000
XOR(1.000000, 1.000000) --> output = 0.007615, expected 0.000000
Save network? (y/n): n

    ~/jules.haquin/NeuralNets | master ?5
```

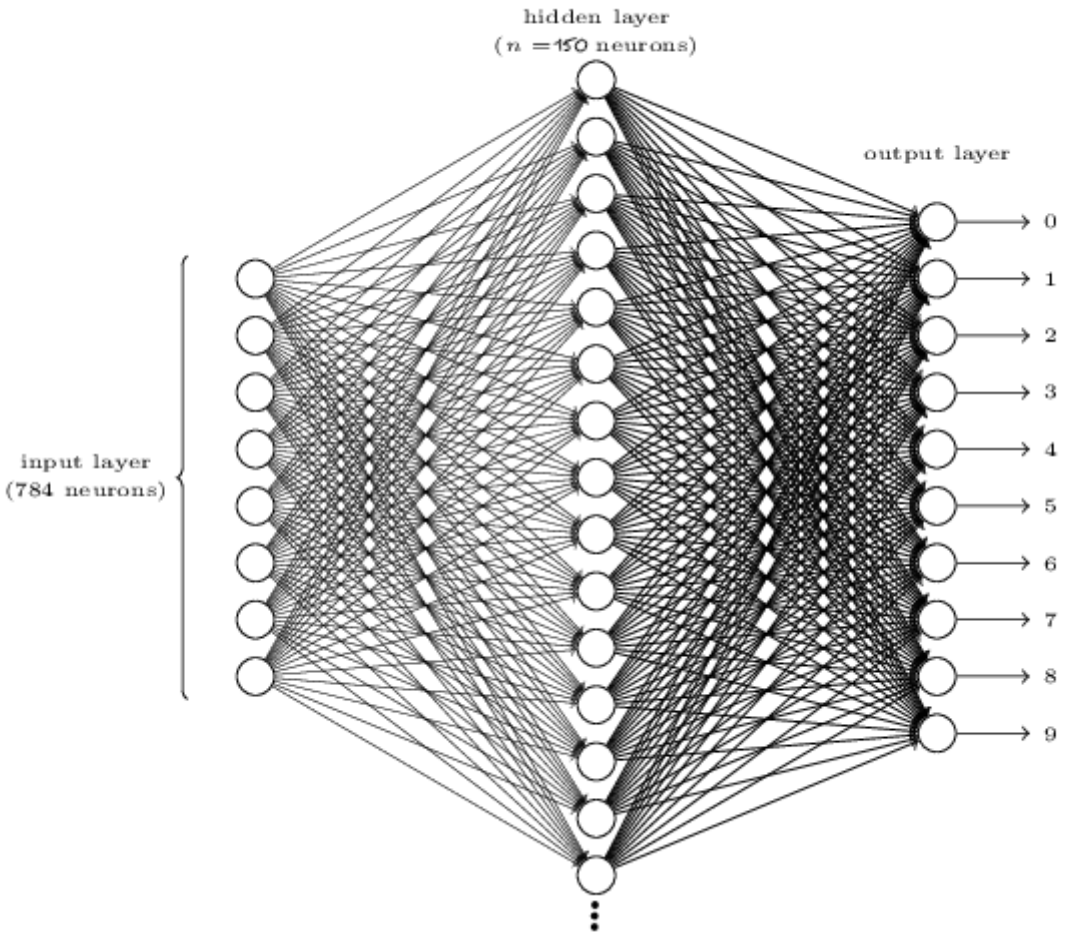
2.4.2 Reconnaissance de chiffres

Pour la reconnaissance de chiffres, notre réseau dispose de 784 neurones d'entrée car il prend des images de 28 pixels sur 28 pixels.

Après de multiples essais, nous avons choisi d'avoir 150 neurones pour la couche cachée. Chaque neurone étant relié à un neurone de la première couche et à un neurone de la couche de sortie.

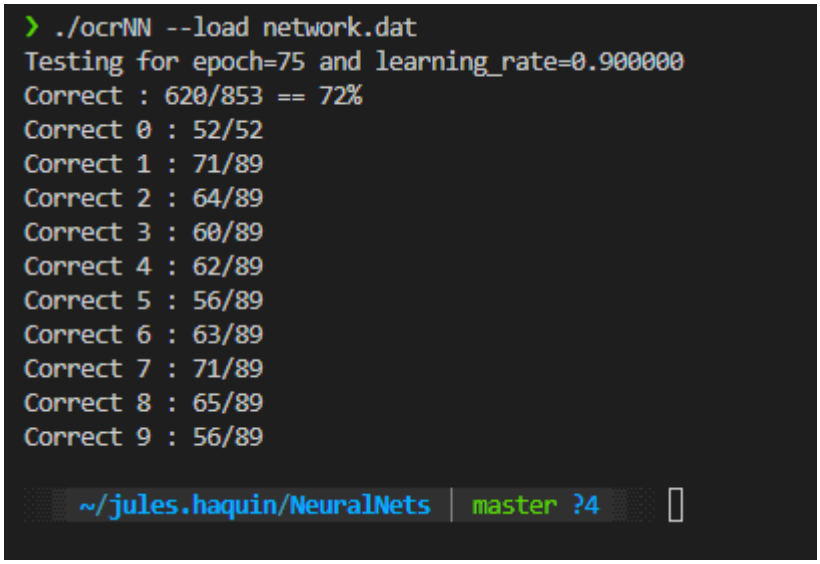
La couche de sortie est quant à elle composée de 10 neurones. Chaque neurone correspond à un chiffre (sauf le zéro qui correspond à une image de case vide). La sortie finale du réseau de neurone est donc égale au neuronne dont la sortie est la plus élevée.

Notre réseau peut être schématisé de la manière suivante :



Pour l'instant les meilleurs résultats que nous avons réussi à obtenir sur un jeu d'image de

test (que le réseau n'a jamais vu pendant son entraînement) sont de l'ordre de 72% de réussite.
Capture d'écran des tests effectués sur un réseau déjà pré-entraîné (les poids et les biais du réseau sont chargés depuis un fichier) :



2.5 Tâches à commencer

2.5.1 Sauvegarde et chargement des poids du réseau de neurones

Pour sauvegarder les poids et les biais du réseau de neurone, nous procédons de la manière suivante :

— la première ligne du fichier est composée d'informations sur le réseau en général comme le nombre d'epochs, la learning rate et le nombre de neurones pour les trois couches
— ensuite nous écrivons tous les poids et tous les biais avec une valeur par ligne

Nous faisons le processus inverse pour charger les informations du réseau de neurones.
Cela se présente en pratique de la manière suivante :

```
❯ ./ocrNN --save exemple.txt
Epoch 0
Epoch 10
Epoch 20
Epoch 30
Epoch 40
Epoch 50
Epoch 60
Epoch 70
Testing for epoch=75 and learning_rate=0.900000
Correct : 620/853 == 72%
Correct 0 : 52/52
Correct 1 : 71/89
Correct 2 : 64/89
Correct 3 : 60/89
Correct 4 : 62/89
Correct 5 : 56/89
Correct 6 : 63/89
Correct 7 : 71/89
Correct 8 : 65/89
Correct 9 : 56/89
❯ ./ocrNN --load exemple.txt
Testing for epoch=75 and learning_rate=0.900000
Correct : 620/853 == 72%
Correct 0 : 52/52
Correct 1 : 71/89
Correct 2 : 64/89
Correct 3 : 60/89
Correct 4 : 62/89
Correct 5 : 56/89
Correct 6 : 63/89
Correct 7 : 71/89
Correct 8 : 65/89
Correct 9 : 56/89

 ~/jules.haquin/NeuralNets | master ?4
```

2.5.2 Jeu d'images pour l'apprentissage

Pour cette partie, nous avons décidé de créer un script en Python qui nous permet de récupérer une multitude de polices avec l'API de Google Fonts.

Nous enregistrons ensuite chaque chiffre de chaque police avec une librairie Python appelée Pillow.

A cela, nous avons ajouté un jeu d'image open-source que nous avons trouvé sur GitHub : Il est accessible **ici**
Nous nous retrouvons donc avec plusieurs milliers d'images pour entrainer notre réseau. Nous avons pris environ 10% de ces images pour constituer un jeu d'image pour tester notre réseau.

3 Conclusion

3.1 Conclusions individuelles

Yann :
Ce début de projet m'a permis de découvrir le traitement d'image et ses complexités. J'ai aussi réussi à m'adapter à certaines spécificités du langage C. J'ai aussi réussi à faire un Makefile fonctionnel pour la suite du projet. Et le traitement d'image a été une découverte intéressante même si remplie de mathématiques et de formules qui ne sont pas ce que je préfère.

Vanina :
Je me suis un peu éparpillée pour ce début de projet mais je saurai être plus efficace pour la suite du projet. J'ai les idées plus claires. J'ai aussi eu du mal a gérer mon temps entre le travail quotidien à EPITA et le projet. Je pense néanmoins finir par réussir malgré les difficultés.

Jules :
Ce projet m'a apporté une connaissance et une compréhension des réseaux de neurones que je n'avais absolument pas avant de commencer.
Il m'a également permis de grandement m'améliorer en C, notamment en manipulant divers notions que nous n'avons pas vu en cours de programmation. Comme par exemple les données de type *struct* ou encore la lecture et l'écriture dans les fichiers.
En somme, je suis satisfait de ce que j'ai pu apporter au groupe et je suis content de ce que ce projet m'a apporté au niveau des connaissances et des compétences.

3.2 Conclusion générale

En conclusion de ce projet, nous avons tous pris en main le C et le projet. Cependant, nous avançons tous à des rythmes très différents. Excepté la détection de la grille et le découpage de l'image, toutes les étapes nécessaires à la première soutenance ont été réalisées. Les réseaux de neurones et le jeu d'image sont déjà quasiment terminés.