

SUDOKU SOLVER

KAZUKI

Io Lart

Anatole Gonnou

Jacques Remy

Axel Cochebin

5 décembre 2021

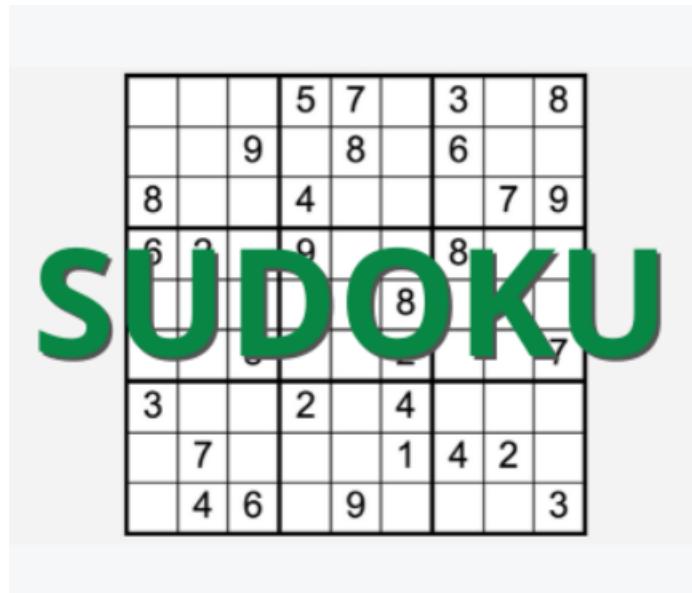


Table des matières

1 KAZUKI	3
1.1 Anatole Gonnون	3
1.2 Axel Cochepin	3
1.3 Jacques Remy	3
1.4 Io Lart	4
2 Traitement de l'image (Io)	5
2.1 Binarisation de l'image	5
2.2 Rotation manuelle de l'image	6
3 Détection et découpage de la grille (Axel)	8
3.1 Découpage grille	8
3.2 Détection grille	8
4 Rotation automatique et programme principal (Axel)	11
4.1 Rotation Automatique	11
4.2 Programme principal	13
5 Solver et affichage (Anatole)	14
6 Réseaux de neurones (Jacques)	16
6.1 Qu'est ce qu'un réseau de neurone ?	16
6.2 La reconnaissance d'images	16
6.3 Base de données MNIST	18
6.4 L'utilisation de MNIST	18
6.5 Les différentes étapes de l'apprentissage	19
6.5.1 Forward Propagation (entrée vers la sortie)	19
6.5.2 Backpropagation (sortie vers l'entrée)	20
6.5.3 Calculer l'erreur du réseau de neurones	20
6.5.4 La fonction sigmoïde	21
6.6 Le résultat et la performance du programme	22
6.7 La reconnaissance de nos propres chiffre	22
7 Répartitions et avancements des tâches	23
7.1 Répartition des tâches	23
7.2 Avancement	23
8 Bonus : Site (Axel)	24
9 Ressentis	24
9.1 Io	24
9.2 Axel	24
9.3 Anatole	24
9.4 Jacques	25
10 Conclusion	26

1 KAZUKI

Notre groupe **KAZUKAKI** est inspiré de la prononciation en japonais de nos initiales. KA pour le A, ZU pour le J et KI pour le I. Il faut savoir que le nom sudoku est né de l'abréviation japonaise de la règle du jeu « *Suji wa dokushin ni kagiru* » (chiffre limité à un seul). Su (chiffre) doku(unique), nous avons donc décidé de reprendre l'étymologie du mot pour créer un nom de groupe.

1.1 Anatole Gonnou

Je suis un passionné de jeux vidéo, en particulier car ils représentent un potentiel de créativité élevée : comme en dessin, on peut créer ce qui nous vient à l'esprit et la seule limite est l'imagination. Très à l'aise en mathématiques, avec un esprit cartésien, le monde de l'informatique me permet de m'exprimer à travers la recherche d'algorithmes efficaces. Les jeux vidéo rétro sont pour moi une parfaite représentation informatique grâce au mélange d'efficacités et d'astuces dans un espace limité et avec un rendu parfait. En Terminale, j'ai choisi la spécialité "Science de l'informatique et du numérique". Cela m'a permis de découvrir la programmation (en particulier Python) et de savoir que je souhaitais faire des études supérieures dans ce domaine. Le projet à réaliser en groupe consiste en l'élaboration d'un jeu de poker. Ce projet m'a donné l'expérience du travail en groupe et de la répartition des tâches en fonction des compétences de chacun.

1.2 Axel Cochepin

J'ai toujours été un grand passionné d'informatique ce qui a fait que je me suis rapidement dirigé vers des études scientifiques et ainsi finalement vers EPITA. Pour commencer, j'ai débuté dans le monde de la programmation avec la découverte de Python au collège et avec la création de petits sites en HTML/CSS. J'ai toujours été plutôt attiré par l'utilisation d'outils informatiques dans tout ce qui touche au traitement d'image avec Photoshop ou la modélisation avec Solidworks et Blender. C'est pour cela que la partie détection et segmentation de la grille, qui fait partie du traitement d'image, a été très passionnante à travailler pour cette première soutenance.

1.3 Jacques Remy

Depuis mon jeune âge, j'ai l'ambition de consacrer ma carrière professionnelle aux métiers du numérique et j'ai rapidement compris que cela passerait par la validation d'un diplôme d'ingénieur. Très tôt sensibilisé à l'informatique, j'ai commencé à apprendre le python et le C++. Par simple curiosité au début, puis naissant un réel engouement et surtout une envie d'en apprendre davantage. Le projet de l'année dernière fut une excellente expérience en laquelle je suis véritablement fier. Nous avons su proposer un jeu au dessus de la hauteur de mes attentes, et je suis sûr que je ne serai pas déçu de ce projet OCR non plus. Bien que celui-ci soit moins libre, je suis tout autant intéressé, et notamment par le réseau de neurone qui sera ma partie lors de cette deuxième soutenance.

1.4 Io Lart

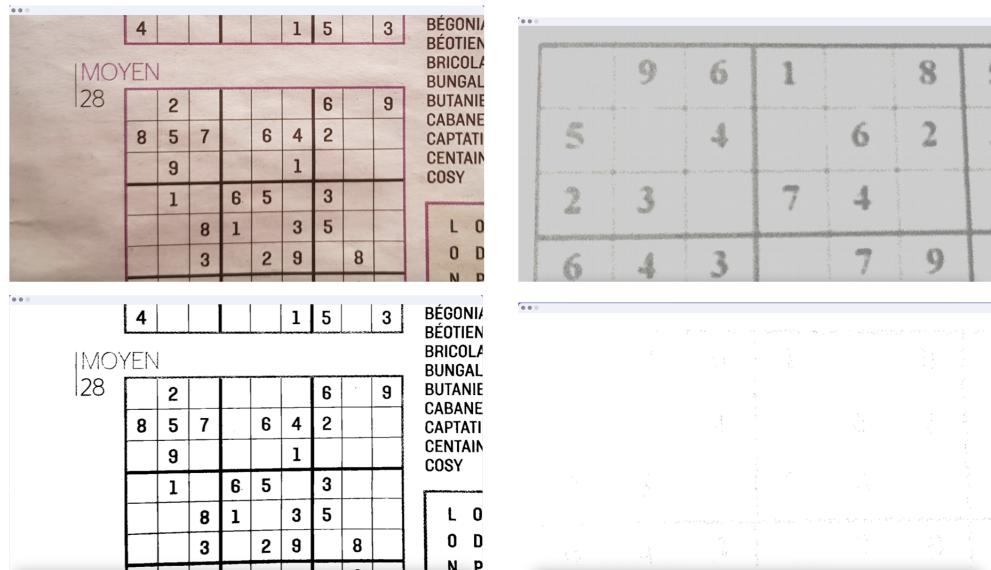
J'ai toujours voulu recevoir un apprentissage scientifique et technique, qui me permettra de participer à la conception, l'innovation et au progrès de notre environnement. La technologie, l'informatique sont des milieux d'avenir. Ils sont attrayants et ne cessent de se développer. Ils permettent de construire le monde de demain et je souhaite y participer. C'est pour cela que l'EPITA est une école qui correspond à mes attentes. De plus, la faible part de fille dans l'ingénierie informatique, m'a davantage poussé à choisir ces études. Je suis curieuse de ce qui m'entoure et l'informatique à ouvert en moi beaucoup de questionnement ainsi que beaucoup d'intérêts. Pour cette première soutenance je me suis occupée du traitement d'image, qui se réalise avant la résolution de la grille. Pour cette soutenance, j'ai réalisé un grand travail de documentation.

2 Traitement de l'image (Io)

2.1 Binarisation de l'image

Pour réaliser la binarisation en noir et blanc de l'image, j'ai commencé par modifier la fonction de notre tp 04. J'ai donc choisi un seuil moyen à 128 (256 valeurs, $256/2 = 128$), toutes les valeurs de pixel supérieurs au seuil se transforment en blanc et celle en dessous, en noir. Après plusieurs tests, je me suis rendue compte que sur pas mal d'images, le rendu n'était pas de bonne qualité.

Comme on peut le voir par exemple sur cette image avec un exemple qui marche, et l'autre non...



C'est là, que le travail de documentation a commencé.

J'ai donc fais des recherches pour binariser efficacement une image en noir et blanc. Je suis tombée dans le domaine du traitement d'image et l'opération correspondante s'appelle la segmentation. J'ai vu qu'il existait différentes méthodes ; le feuillage, la classification, le clustering...

D'après la première réunion sur le projet, j'avais le souvenir d'avoir entendu parler de la méthode d'otsu, méthode, qui après renseignement, m'a parût très efficace, mais assez complexe à comprendre.

Il existe donc deux méthodes deux seuillages différentes (passer d'une image en niveau de gris en image binaire (noir et blanc)), le feuillage manuel et celui automatique qui correspond à la méthode d'otsu.

La méthode d'otsu consiste à séparer les pixels en deux catégories ; celle des pixels noirs et celle des pixels blancs, avec un k qui sépare les deux classes différentes. Le but de cette méthode est de trouver le k qui sépare au mieux le fond et les objets de l'image à traiter.

Pour calculer k, on va trouver la variance interclasse entre la catégories des pixels noirs et celle des pixels blancs, pour tous k possible entre 0 et 255 (les valeurs de pixels).

K sera donc le seuil et il représente, la plus haute variance interclasse.

J'ai pu déchiffrer des étapes nécessaires à cette méthode qui correspondent plus au moins aux différentes fonctions :

- On met l'image en niveau de gris, et on construit un histogramme de ses niveaux de gris.
- Normaliser l'histogramme (obtenir un histogramme dont toutes les valeurs sont comprise entre 1 et 0.). On obtient donc une distribution de probabilités en fonction de chaque valeur de pixel.
- Calcul de la probabilité qu'un pixel de l'image appartienne à la catégorie des pixels noirs. Cela se fait en calculant la somme de toutes les colonnes de l'histogramme comprises entre 0 et k.
- Calcul de la probabilité qu'un pixel de l'image appartienne à la catégorie des pixels blancs. En faisant simplement $1 - \text{proba}(\text{des pixels noirs})$ calculée dans l'étape précédente.
- Calcul de la variance interclasses entre la catégorie des pixels blancs et celle des pixels noirs
- Toutes les étapes sont répétées pour toutes les valeurs de k possibles.

J'ai donc réalisé cette méthode en faisant les fonctions associées aux différentes étapes.

Après exécution de la fonction, je me suis rendue compte que la méthode otsu ne marchait pas sur beaucoup d'images et que le simple noir et blanc était plus efficace sur une partie des images. Je me demande donc qu'est ce que j'ai pu mal faire.

J'ai réalisé une fonction qui supprime le bruit, pour pouvoir l'exécuter avant la méthode otsu.

Je n'ai pas réussi à faire un cas général efficace pour chaque image.

2.2 Rotation manuelle de l'image

En ce qui concerne la rotation de l'image, j'ai utilisé plusieurs formules de trigonométrie.

Il faut bien distinguer l'image d'entrée et celle de sortie, ce qui m'a un peu pris de temps avec les librairies SDL.

La taille du fond de l'image de sortie sera supérieure à celle d'entrée, pour la calculer, j'ai utilisé cette formule :

$$X = x * \cos(\text{angle}) + y * \sin(\text{angle}) \quad Y = y * \cos(\text{angle}) - x * \sin(\text{angle})$$

J'ai utilisé cette formule pour convertir les degrés en radians :

$$\text{radian} = 2\pi * \text{degree} / 360$$

Et j'ai ensuite pris chaque pixel de l'image de sortie et on calcule le pixel source correspondant de l'image d'entrée.

Pour la fonction automatique de rotation, j'ai laissé Axel la faire. Il c'était occupé des algorithmes de détection, ce qui facilite l'algorithme pour la rotation automatique.

Voici quelques exemples de la fonction rotation :

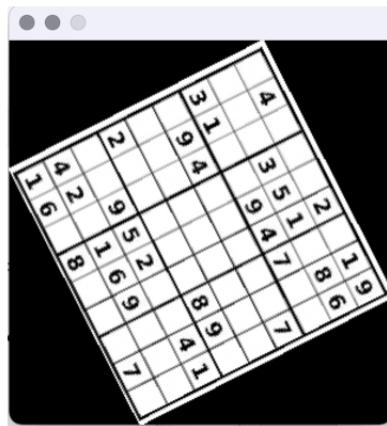
rotation de -180°

4			2	1	9			
		3	5	1	8	6		
3	1		9	4	7			
	9	4				7		
2				8	9			
	9	5	2			4	1	
4	2	1	6	9				
1	6	8			7			



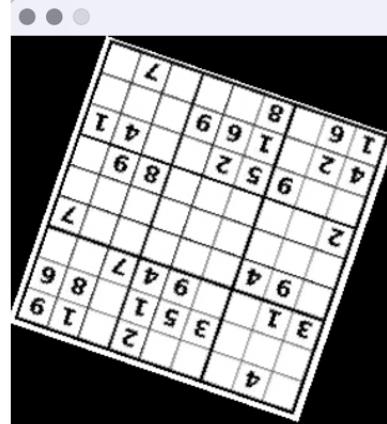
rotation de 63°

4			2	1	9			
		3	5	1	8	6		
3	1		9	4	7			
	9	4				7		
2				8	9			
	9	5	2			4	1	
4	2	1	6	9				
1	6	8			7			



rotation de 559°

4			2	1	9			
		3	5	1	8	6		
3	1		9	4	7			
	9	4				7		
2				8	9			
	9	5	2			4	1	
4	2	1	6	9				
1	6	8			7			



3 Détection et découpage de la grille (Axel)

3.1 Découpage grille

Dans un premier temps, nous avons développé un programme permettant de segmenter l'image uniquement si elle était droite. Pour résoudre les problèmes d'orientation de l'image et la possibilité que l'image ne soit pas droite, nous avons développé un programme de rotation automatique, comme nous allons le voir prochainement. Ainsi, comme précédemment, le programme découpe l'image en 81 cases et les enregistre une par une dans un dossier. Ces images pourront ensuite être traiter par le programme de machine learning qui va détecter chaque chiffre un par un. Le programme récupère donc les dimensions de la grille, les divisent en 9 pour le nombre de cases, et les parcours une à une pour les enregistrer. Les cases sont donc enregistrées au format "47", avec 4 la colonne et 7 la ligne.

3.2 Détection grille

Au début du projet, nous nous étions dirigés vers la méthode "Hough Transform" pour réaliser la détection de la grille. Nous avions ainsi commencé à réaliser la détection par cette méthode, et avons donc programmé la possibilité d'appliquer un flou Gaussien à l'image.



Image d'origine

Filtre Flou gaussien appliqué

Mais après réflexions, nous avons réalisé que cette méthode utilisée de nombreuses étapes de traitement d'image qui ne sont pas forcément nécessaires à la détection de la grille, vu qu'une phase de traitement de l'image est déjà faite en amont. Nous avions ainsi décidé de partir sur une méthode par "histogramme", vu que cette méthode convient mieux au traitement de l'image réalisé en amont. Ainsi, nous allons pouvoir utiliser la propriété de l'image qui est d'être binarisé pour réaliser d'abord deux histogrammes. Nous aurons donc un histogramme pour la détection des colonnes et un autre pour les lignes. Chaque histogramme serait créé de la manière suivante :

Pour chaque ligne et colonne :

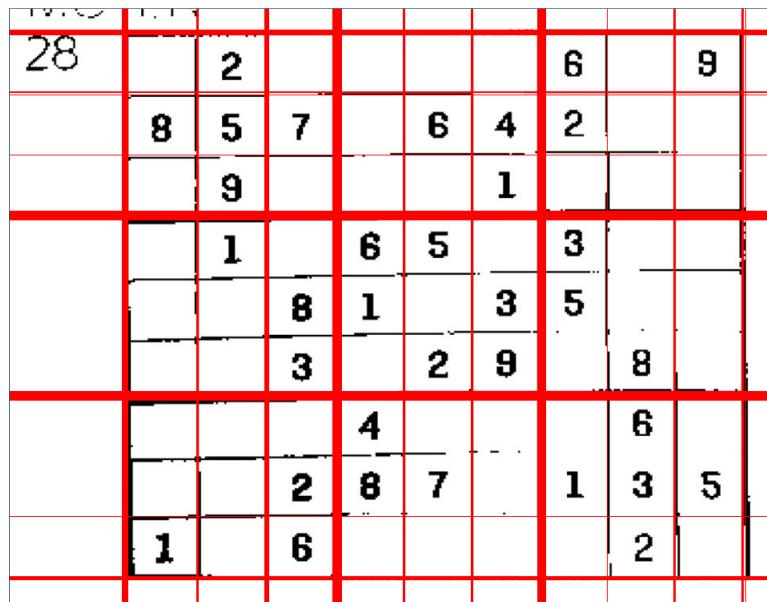
- Pour chaque pixel, on regarde s'il est noir ou blanc
- S'il est noir, on l'ajoute à la somme des pixels noirs de la ligne/colonne
- Si on tombe sur un pixel blanc, on regarde si la somme dépasse le maximum déjà rencontré à la suite de pixel noir et si c'est le cas, on met la somme au maximum et la somme à 0
- Puis, si on dépasse la longueur/largeur divisée par 4 alors on ajoute la somme à l'histogramme, sinon on met 0

Ensuite, nous allons utiliser chaque histogramme pour trouver l'emplacement de la première et dernière colonne et ligne, et ainsi avoir la largeur et la longueur de la grille en réalisant une soustraction. Avec ces données, on peut donc rogner l'image pour récupérer la grille et ainsi la découper.

28

	2				6		9
8	5	7		6	4	2	
	9				1		
	1		6	5		3	
		8	1		3	5	
	3		2	9		8	
			4			6	
		2	8	7		1	3
1	6					2	

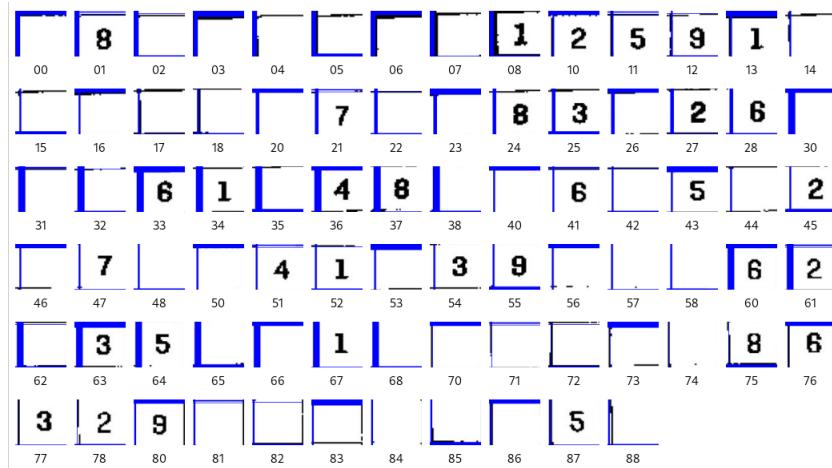
Comme vous pouvez le voir ci-dessus, pour la première étape nous récupérons une image binarisé, bien rotationné et relativement bien cadré sur la grille.



Ensuite, avec l'image précédente, nous allons donc détecter les lignes et les colonnes. Pour cela, nous allons remplacer tous les pixels de chaque ligne ou colonne détectées par des pixels rouge.

	2					6		9
8	5	7		6	4	2		
	9				1			
	1		6	5		3		
		8	1		3	5		
	3		2	9		8		
			4				6	
		2	8	7		1	3	5
1		6					2	

Avec cette nouvelle image et les deux histogrammes remplies nous n'avons plus qu'à nous en servir pour isoler la grille, comme ci-dessus.

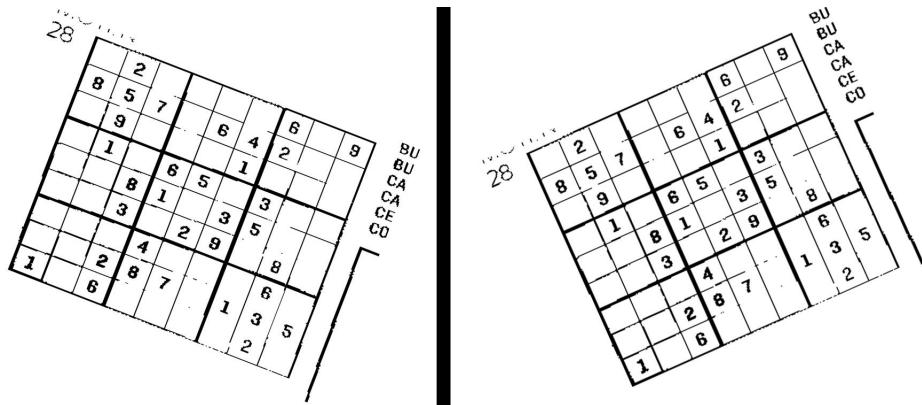


Finalement, après avoir isolé la grille nous n'avons plus qu'à appliquer le découpage pour isoler les cases et les enregistrer dans un dossier pour leur futur traitement.

4 Rotation automatique et programme principal (Axel)

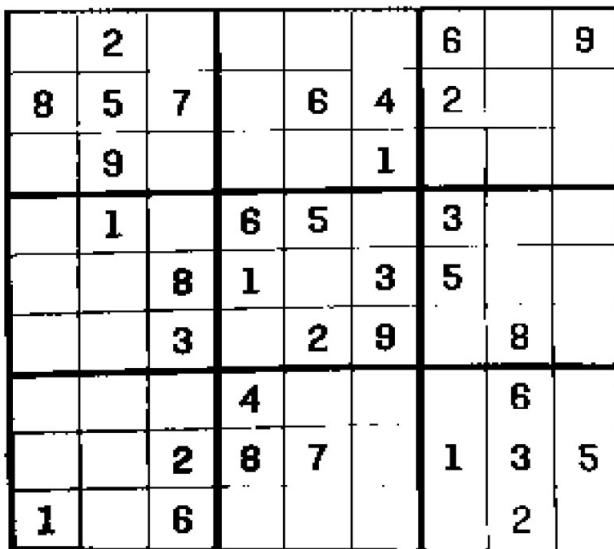
4.1 Rotation Automatique

Tout d'abord, pour la rotation automatique, nous avons importé la fonction de rotation que Io avait faite et la fonction d'Axel permettant la détection et la segmentation d'une grille. Premièrement, nous avons fait une fonction permettant de compter le nombre d'éléments dans un histogramme, et ainsi le nombre de colonnes et de ligne détecté. Cette fonction va ainsi nous être utile dans la fonction permettant la rotation automatique, la détection et la segmentation d'une image. Nous avons donc une fonction qui avec le nom d'une image et un entier définissant dans quelle sens l'on doit effectuer la rotation, nous segmentent les 81 cases du Sudoku dans un dossier. On va donc initialiser deux histogrammes, un pour la largeur, l'autre pour la longueur, et un angle de rotation que l'on met à 0 degré. Puis on lance une boucle qui va effectuer autant de rotation nécessaire dans le sens défini jusqu'à trouvé un nombre cohérent de ligne et de colonne grâce à la fonction permettant de trouvé le nombre d'élément d'un histogramme. Une fois la condition remplie, on sort de la boucle, cela veut dire que l'image est maintenant droite et donc que l'on peut effectuer la détection et la segmentation de la grille.



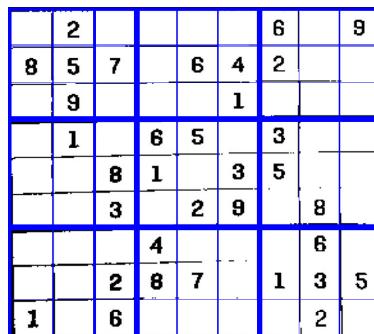
On prend donc soit une image qui est inclinée sur la gauche, soit incliné sur la droite que l'on va donner au programme.

28

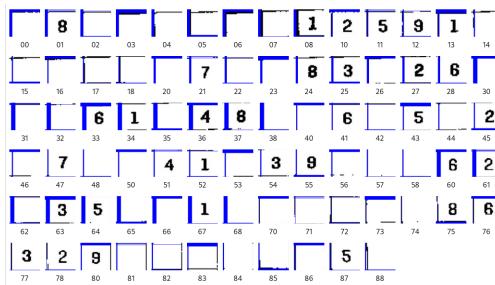


BU
BU
CA
CA
CE
CO

On a donc ensuite une image droite que nous allons donner au programme de découpage et de segmentation.



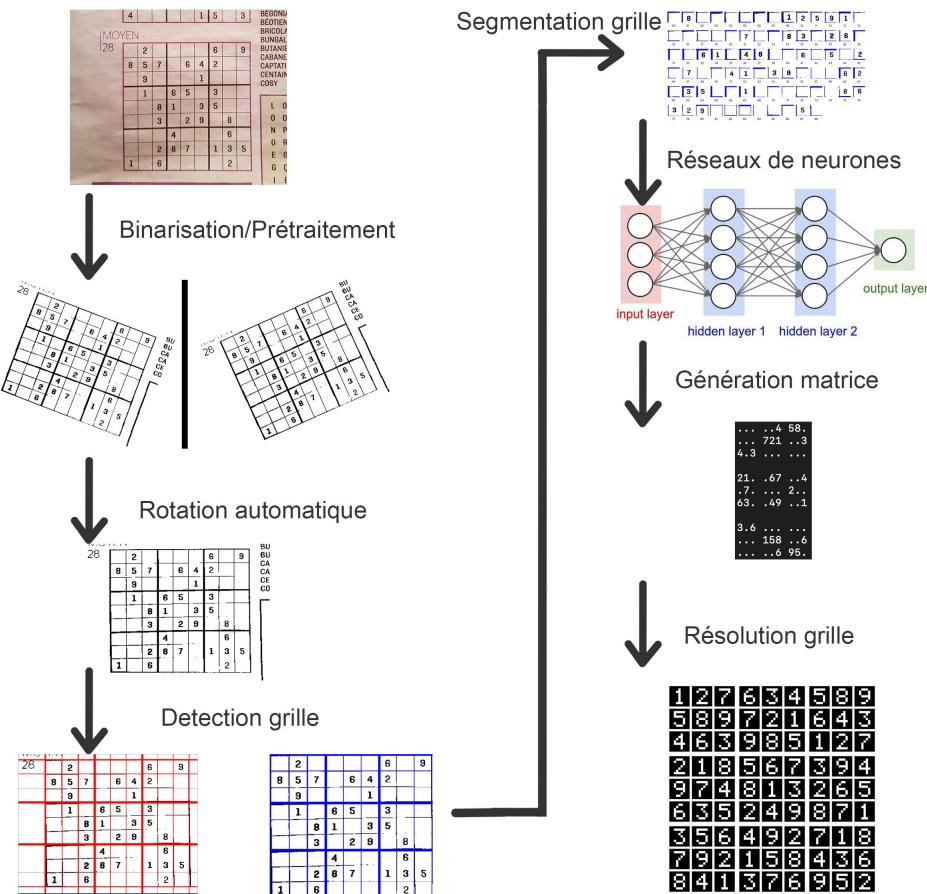
Nous avons ensuite donc la grille détecté avec ses lignes et colonnes.



Pour finalement avoir le dossier "Cases" rempli de toutes les cellules de la grille.

4.2 Programme principal

Nous allons maintenant donc voir la mise en commun des différentes étapes de la résolution du sudoku. Ainsi, premièrement, nous allons binariser l'image donnée et réaliser un pré traitement de l'image. Puis nous allons réaliser une rotation automatique sur la gauche avec une détection et une segmentation. On lance le programme de machine learning pour reconstruire une matrice représentant la grille de sudoku. On utilise le solver pour résoudre la grille retrouvé et si la fonction retourne un "faux" alors on relance les fonctions précédentes sur l'image de base, mais avec une rotation vers la droite. Et finalement on affiche la grille résolu.



5 Solver et affichage (Anatole)

La consigne était de résoudre un sudoku présenté en format texte. Nous devons programmer un lecteur de texte qui nous renvoie une matrice représentant le sudoku. Donc, ce programme ouvre le fichier. Si celui-ci n'existe pas le programme envoie une erreur "parser : le fichier n'existe pas". Dans le fichier, il y a des chiffres et des points qui représentent les cases vides. Le programme examine chaque caractère jusqu'à la fin, vérifie la présence d'un chiffre ou d'un point et les placent dans la matrice à la bonne position. Les points sont des 0 dans la matrice. Si c'est un caractère autre qu'un chiffre ou un espace ou un saut de ligne ou un chiffre mal placé, ou si le fichier contient plus de caractères que nécessaire, le programme nous renvoie une erreur "parser : le fichier pas au bon format".

Ceci est un exemple de fichier texte qui représente un grille de sudoku non résolu

...	..4	58.
...	721	..3
4.3
21.	.67	..4
.7.	...	2..
63.	.49	..1
3.6
...	158	..6
...	..6	95.

Dans cette partie, nous allons voir comment nous avons résolu le sudoku en question. Pour représenter le sudoku nous faisons une matrice à 2 dimensions de taille 9x9.

Dans cette matrice « grid », les cases vides du sudoku, les cases vides sont représentées par « 0 », les chiffres sont indiqués par leur valeur entre 1 et 9. La fonction « solveur » a pour argument la matrice et celle-ci appelle une fonction recursive et celle-ci prend comme arguments la matrice et deux chiffres (x,y) qui représentent les coordonnées des cases de la matrice. On appelle la fonction récursive avec les coordonnées (0,0). Cette fonction récursive renvoie un booléen. La fonction avance de gauche à droite et de haut en bas. Elle regarde le chiffre à la position de (x,y) de la matrice Si ce chiffre n'est pas 0, on incrémente x de 1 et lorsque x=9, on met x à 0 et on incrémente y de 1. On appelle la fonction récursive avec ces nouvelles coordonnées. Si le chiffre est 0, on prend 1, on regarde si le 1 ne figure ni dans la même ligne, ni dans la même colonne, ni dans le carré 3x3 et on le place dans la grille, on appelle avec la fonction récursive, et si la fonction récursive renvoie « faux », on essaye la même méthode avec 2, jusqu'à 9. Même si 9 ne peut pas être placé, la fonction retourne « faux ». La fonction se termine quand y=10 et retourne « vrai ». Cette méthode s'appelle le Backtracking (brut force)

Après avoir résolu le sudoku, il faut que nous sauvegardions le résultat dans un fichier texte. Pour cela, on crée un fichier texte et on écrit les chiffres de la matrice dans le même format que le fichier de base, en remplaçant les points par les nouveaux nombres.

Ceci est un exemple de fichier texte qui représente un grille de sudoku résolu

1	2	7	6	3	4	5	8	9
5	8	9	7	2	1	6	4	3
4	6	3	9	8	5	1	2	7
2	1	8	5	6	7	3	9	4
9	7	4	8	1	3	2	6	5
6	3	5	2	4	9	8	7	1
3	5	6	4	9	2	7	1	8
7	9	2	1	5	8	4	3	6
8	4	1	3	7	6	9	5	2

En bonus pour cette première soutenance, nous avons décidé de créer une image qui représente la matrice. Pour cela nous utilisons la bibliothèque SDL. Cette bibliothèque permet de créer des images ou des fenêtres. Nous avons décidé de créer nous-mêmes un programme qui dessine des chiffres. Nous créons pour chaque chiffre une image de base qui est noire. Chaque chiffre est représenté par une liste d'entiers. Ces entiers représentent une ligne de l'image du chiffre. Cet entier en base 2 permet de savoir quand afficher la couleur blanche. Nous parcourons bit à bit cet entier, le pixel blanc est affiché lorsque nous rencontrons un 1. Nous avons créé une fonction qui construit les traits de la grille de sudoku. Nous avons dû créer une fonction qui permet de créer des carrés de couleur blanche pour éviter de construire une image de petite taille. Cela permet d'obtenir une image assez grande.

deux image de la grille de sudoku de taille different

1	2	7	6	3	4	5	8	9
5	8	9	7	2	1	6	4	3
4	6	3	9	8	5	1	2	7
2	1	8	5	6	7	3	9	4
9	7	4	8	1	3	2	6	5
6	3	5	2	4	9	8	7	1
3	5	6	4	9	2	7	1	8
7	9	2	1	5	8	4	3	6
8	4	1	3	7	6	9	5	2

1	2	7	6	3	4	5	8	9
5	8	9	7	2	1	6	4	3
4	6	3	9	8	5	1	2	7
2	1	8	5	6	7	3	9	4
9	7	4	8	1	3	2	6	5
6	3	5	2	4	9	8	7	1
3	5	6	4	9	2	7	1	8
7	9	2	1	5	8	4	3	6
8	4	1	3	7	6	9	5	2

6 Réseaux de neurones (Jacques)

La reconnaissance d'image est de plus en plus utilisée de nos jours, que ce soit par exemple pour déverrouiller son téléphone ou encore pour faire rouler des voitures autonomes. Dans cette partie, nous allons nous intéresser à ce qu'est la reconnaissance d'image et comment cela fonctionne, puis nous verrons de manière plus approfondie comment créer un programme de reconnaissance d'image.

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

Le réseau de neurones est un élément de l'OCR très important car en effet c'est lui qui sera capable d'identifier les chiffres afin de réaliser le sudoku. Cette partie est compliquée à mettre en place car le réseau agit comme un cerveau : il est capable d'apprendre pour se perfectionner et ainsi avoir des résultats de plus en plus proches de la vérité.

Les réseaux de neurones sont constitués de plusieurs couches (au minimum deux) :

- couche d'entrée : les neurones de cette couche contiendront l'information que l'on souhaite tester
- couche de sortie : ces neurones nous donneront le résultat du problème que le réseau aura résolu
- une ou plusieurs couche(s) cachée(s) : dans cette couche les neurones serviront seulement pour le calcul, leurs valeurs ne seront exploitées que par le réseau de neurones. Pour notre part, nous avons décidé de réaliser un réseau avec deux neurones dans sa couche cachée.

De plus, chaque connexion entre deux neurones possède un poids qui permet de faire les différents calculs ainsi qu'une valeur de sortie (comprise entre 0 et 1) qui est la valeur à tester pour la couche d'entrée et le résultat d'un calcul pour les autres couches, cette valeur correspond à l'information que chaque neurone transmet aux neurones suivants.

6.2 La reconnaissance d'images

Prenons tout d'abord un cas très simple, où l'on demande à notre programme de reconnaissance d'image de différencier les chiffres de 0 à 9 écrits à la main.



Il est facile pour nous, humains, de différencier ces trois chiffres, respectivement : trois, cinq et huit. Cela semble évident car nous avons appris à analyser les informations que notre rétine envoie à notre cerveau dès notre plus jeune âge, et nous pouvons même reconnaître des images bien plus complexes comme par exemple des visages. Mais pour apprendre à un ordinateur à reconnaître des images, il va nous falloir comprendre de manière plus approfondie ce qu'est réellement la reconnaissance d'image. Prenons ces trois autres chiffres écrits à la main :



La valeur des pixels des images ci-dessus est différente de la valeur des pixels des images présentées plus haut, alors qu'elles représentent les mêmes chiffres. Il existe effectivement un nombre immense de manières d'écrire un trois ou un cinq. Mais d'une certaine façon, notre cerveau parvient à interpréter ces différentes combinaisons de pixels comme étant le même chiffre et de reconnaître d'autres images comme étant des chiffres différents. De la même manière, un ordinateur devra calculer une combinaison des pixels d'une image pour reconnaître ce qu'elle représente. Pour cela, imaginons que les pixels d'une image en noir et blanc ne sont en réalité que des nombres rationnels compris entre 0 et 1. Un pixel noir est représenté par le chiffre 1 et un pixel blanc par 0. Un pixel gris foncé aura par exemple une valeur de 0.88. Une des manières les plus simples pour entraîner notre réseau de neurones est d'utiliser une base de données de chiffres manuscrits.

6.3 Base de données MNIST

La base de données MNIST pour *Modified or Mixed National Institute of Standards and Technology*, est une base de données de chiffres écrits à la main. La base MNIST est devenue un test standard¹. Elle regroupe 60000 images d'apprentissage et 10000 images de test.

7	1	0	6	2	0	6	2
3	8	0	0	8	8	4	7
5	8	7	3	9	0	5	8
1	5	0	2	8	4	2	3
0	4	3	9	8	2	1	8
5	0	1	6	6	5	5	2
1	7	7	1	2	3	7	3
6	3	7	6	0	1	4	0

6.4 L'utilisation de MNIST

Afin de pouvoir entraîner notre réseau, nous avons à notre disposition une base de données de 6000 images. IL reste à savoir comment l'utiliser pour entraîner notre réseau. Tout d'abord, il faut savoir que la base de donnée se découpe en 2 morceaux : l'image et le label.

- l'image est un fichier regroupant chaque chiffres écrits en hexadécimal.
- le label est le nom du chiffre de chaque image. Par exemple, pour une image avec un neuf dessiné, le label retournera "9".

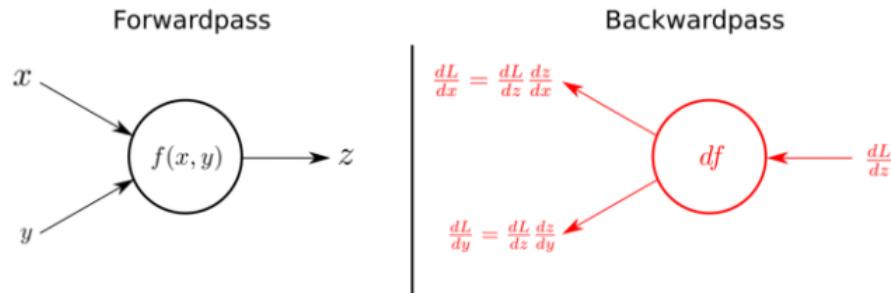
Le but sera de modifier les neurones de notre réseau si le résultat de notre premier *Feed Forward* ne correspond pas au label de l'image en entrée.

6.5 Les différentes étapes de l'apprentissage

Le but de l'apprentissage est de permettre au réseau de neurone de s'améliorer. Cela consiste simplement à modifier les poids de chaque connexion jusqu'à ce qu'elles atteignent leurs valeurs optimales, mais cela nécessite plusieurs étapes pour parvenir à obtenir les nouvelles valeurs de poids (que l'on traitera par la suite). Il est aussi possible d'agir sur la vitesse de l'apprentissage en rajoutant plus de neurones, en connectant les neurones d'entrée directement à ceux de sortie ou encore en rajoutant des neurones de biais (des neurones qui ne sont pas connectés qu'à la couche suivante mais pas à la précédente). Le pas d'apprentissage est une valeur qui permet de régler la vitesse à laquelle le réseau apprend mais aussi la précision de l'apprentissage. Plus le pas sera grand, plus l'apprentissage sera rapide mais moins précis, et inversement. Il existe différentes méthodes directes pour qu'elle soit le plus précis possible comme : le *Time-Based Decay*, le *Step Decay* ou bien l'*Exponential Decay*.

6.5.1 Forward Propagation (entrée vers la sortie)

La première étape par laquelle passe notre réseau de neurone est l'algorithme de calcul en avant (*Forward Propagation*). Cette algorithme permet de trouver les valeurs de sortie du réseau de neurones. La première étape est de calculer une valeur d'activation pour chacun des neurones. Cette valeur se calcule grâce aux poids placés sur les connexions, elle correspond à la somme des poids des connections multipliés par la sortie des neurones correspondants de la couche supérieure. Cette valeur est ensuite appliquée à une fonction qui donnera une valeur de sortie. Cette fonction est la fonction sigmoïde que nous verrons par la suite.



6.5.2 Backpropagation (sortie vers l'entrée)

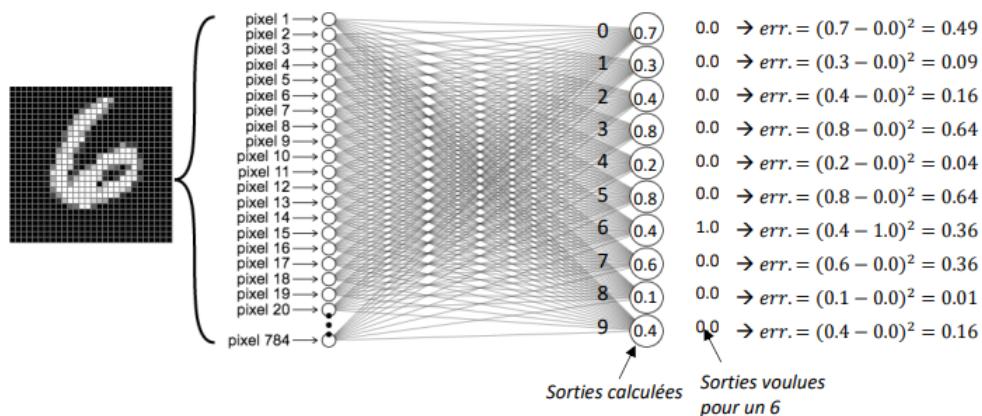
la rétro-propagation du gradient est une méthode pour entraîner un réseau de neurones en mettant à jour les poids de chaque neurone de la dernière couche vers la première. Pour pouvoir appliquer cet algorithme il va falloir que le réseau connaisse la valeur qu'il doit obtenir ce qui permettra de trouver l'erreur et permettre de mettre 'à jour les poids. Le processus d'apprentissage se passera en cinq étapes :

- Tout d'abord il faut effectuer le calcul en avant pour connaître toutes les sorties du réseau (*Forward Propagation*)
- Ensuite sur les neurones de sortie on calcule une valeur d'erreur, en connaissant leurs valeurs de sortie ainsi que celles qu'ils doivent avoir (valeur théorique connue grâce aux labels).
- Ensuite on met à jour le poids de chaque connexion allant vers les neurones de sortie.
- On calcule l'erreur sur la couche cachée précédant celle que l'on vient de corriger.
- Enfin, On met à jour les poids des connexions allant sur la couche caché.

On répète les pas 1 à 6 pour les autres exemples à apprendre. L'ensemble de ces répétitions se nomme une itération. Après une itération, la sortie du réseau sera un peu plus proche de la bonne solution. L'algorithme « *backprop* » consiste à effectuer plusieurs itérations jusqu'à ce que l'erreur soit suffisamment petite.

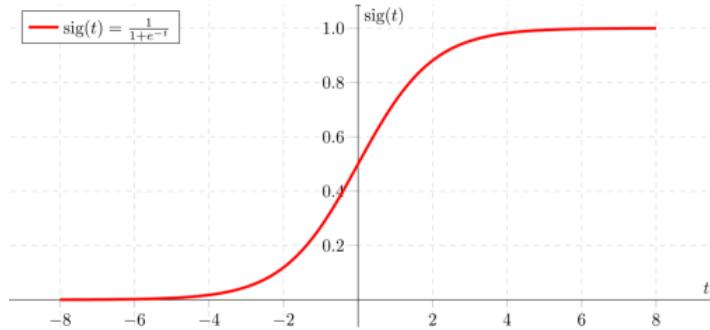
6.5.3 Calculer l'erreur du réseau de neurones

Pour calculer cette erreur, on donne à notre ordinateur une image de chiffre écrit à la main et les 10 sorties correspondante à cette image, c'est-à-dire un 1 pour la bonne sortie et des 0 pour les 9 autres. On calcule les 10 sorties avec les weights et les pixels de l'image puis on compare ces sorties calculées avec celle que l'on aurait dû obtenir. Le but va donc être de réduire la différence qu'il y a entre les sorties calculées et les sorties correctes.



6.5.4 La fonction sigmoïde

Dans la figure ci-dessus, les sorties affichées (dans les cercles) sont toutes comprises entre 0 et 1. Ce n'est pas un hasard. Cela vient du fait qu'après avoir multiplié et additionné tous ces pixels et ces poids ensemble, les résultats obtenus sont encore évalués par la fonction sigmoïde :



Cette fonction transforme tous les nombres compris dans l'intervalle $]0; 1[$. Ainsi, si un résultat est très grand, la sortie sera proche de 1 et s'il est très négatif, alors la sortie s'approchera de 0. Vous vous demandez peut-être à quoi sert cette fonction sigmoïde. Le but de la fonction sigmoïde est de faire que tous les résultats obtenus se situent entre 0 et 1. Une sortie approchant 0 signifie que le filtre ne correspond pas à l'image évaluée et plus on s'approche de 1, plus il y a de chances que le filtre corresponde à l'image affichée. Une sortie valant plus de 1 ou moins de 0 n'aurait donc aucun sens. De plus, les données que l'on fournira par la suite à l'ordinateur pour qu'il apprenne à différencier les images sont des vecteurs contenant des 0 et un 1 pour la bonne sortie. Notre but sera de faire que les sorties calculées ressemblent aux données fournies à l'ordinateur. En utilisant la fonction sigmoïde, les nombres que l'on fait passer à travers celle-ci doivent soit être très grand ou soit très petit pour se rapprocher de 1 ou 0, mais ils ne doivent pas être des valeurs exactes. Les sorties calculées correspondront donc mieux aux données fournies. En soit notre réseau pourrait fonctionner sans la fonction sigmoïde, mais il fonctionne mieux avec.

6.6 Le résultat et la performance du programme

Grâce aux tests de la base de donnée MNIST, nous pouvons désormais calculer le pourcentage de performance de notre programme grâce au calcul :

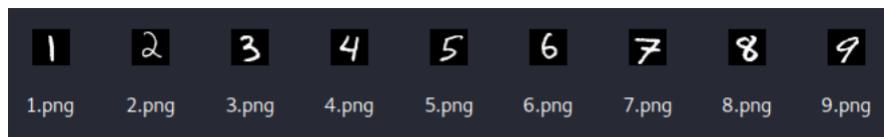
$$\frac{\text{Nombre d'images - Compteur d'erreur}}{\text{Nombre d'images}} \times 100$$

Ici, le compteur d'erreur est un simple compteur que l'on incrémente lorsque le résultat ne correspond pas au label de l'image d'entrée. Voici ce que la fonction nous renvoie :

```
[root@kali]~/Desktop/Projets3/OCRSudokuS3/NeuralNetwork]
# ./nn.out 2
Test Accuracy: 8.86%
Training epoch 1/5
Test Accuracy: 93.41%
Training epoch 2/5
Test Accuracy: 95.07%
Training epoch 3/5
Test Accuracy: 95.76%
Training epoch 4/5
Test Accuracy: 96.02%
Training epoch 5/5
Test Accuracy: 96.17%
```

6.7 La reconnaissance de nos propres chiffre

Tout d'abord, afin de récupérer nos chiffres, il faut redimensionner l'image en 28x28 car c'est la dimension de notre base de donnée servant à entraîner le réseau de neurones. Puis, nous devons récupérer chaque pixel de l'image d'entrée pour les rentrer dans les pixels de notre structure image. Avec les pixels de notre image entrées dans notre programme, nous pouvons désormais (après avoir entraîner notre réseau) exécuter un *feedForward* sur celles-ci et récupérer la valeur de sortie.



```
[root@kali]~/Desktop/Projets3/OCRSudokuS3/NeuralNetwork]
# ./nn.out 3
1 2 3 4 5 6 7 8 9
```

Comme on peut le voir ici, les images d'entrée du réseau ne sont pas les images des chiffres découpés sur le sudoku. En effet, lorsque l'on y rentre ces images, celui-ci ne renvoie qu'une partie des chiffre mais le reste sont des nombres plus grand que 10 mais nous n'avons pas réussi à résoudre ce problème.

7 Répartitions et avancements des tâches

7.1 Répartition des tâches

Tâches	Anatole	Jacques	Axel	io
Chargement d'une image				x
Suppression des couleurs				x
Prétraitement				x
Détection de la grille			x	
Détection des cases de la grille			x	
Rotation Automatique de la grille			x	
Réseau de neurones		x		
Reconstruction de la grille	x			
Résolution de la grille	x			
Affichage de la grille résolue	x			
Sauvegarde de la grille résolue	x			

7.2 Avancement

Tâches	Soutenance Finale
Chargement d'une image	100%
Suppression des couleurs	100%
Prétraitement	90%
Détection de la grille	90%
Détection des cases de la grille	100%
Rotation Automatique	100%
Réseau de neurones	80%
Reconstruction de la grille	100%
Résolution de la grille	100%
Affichage de la grille résolue	80%
Sauvegarde de la grille résolue	100%

8 Bonus : Site (Axel)

Lien : <https://thefoxell.github.io/SudokuSite/index.html>

Nous avons donc réalisé un site présentant :

- Notre projet
- Les différents membres du groupe
- Les logiciels utilisés
- L'école

Et un onglet permettant de télécharger le projet

9 Ressentis

9.1 Io

J'ai le sentiment d'avoir appris pas mal de choses, notamment une petite partie sur le traitement d'image qui était un domaine encore peu connu pour moi. Et j'ai été assez époustouflé des techniques existantes et efficaces sur le traitement d'image. Cependant, je suis assez déçue du résultat de la méthode d'otsu. Je ne suis pas arrivée à un résultat assez efficace pour les types d'images différentes. J'ai été un peu démoralisée car je me suis quand même assez acharnée sur cette méthode.

9.2 Axel

Par la réalisation de ce projet, j'ai pu en apprendre beaucoup sur les éléments nécessaires à la conception d'un OCR. Les parties que j'ai eues à traiter m'ont très intéressé, notamment par la création de méthodes permettant de répondre aux problématiques de chaque partie. Grace à ce projet, j'ai aussi pu en apprendre grandement sur le C, la gestion de liste, matrice, la bibliothèque SDL, mais aussi évidemment à travailler sur un projet de groupe.

9.3 Anatole

Je trouve ce projet est plus intéressant que les années précédent. Fait les fonction m'amuse assez et je découvre des méthodes extraordinaires. Il y a une bonne ambiance dans le groupe. Dessiner pixel par pixel m'a beaucoup amuser à faire. Prochaine je dois être plus accès pour mes camarades pour les aider rapidement en cas de problème trouvez quoi faire dans la suite du projet comme faire l'interface graphique du programme.

9.4 Jacques

La deuxième de ce projet à été encore plus enrichissante que la première. En effet, la réalisation du réseau de neurone sur la fonction XOR à déjà été une véritable expérience pour moi, mais réussir à en implémenter une qui s'occupe de traiter des images et de les reconnaître aurait été inimaginable pour le moi du début d'année. Grâce à l'infinité de documentations sur internet, j'ai pu en apprendre énormément sur le fonctionnement d'un réseau de neurone et notamment son implémentation sur des images. Le travail de recherche à été l'étape principale pour cette partie : il me fallait comprendre comment réaliser ce réseau. C'est pour cela que je m'y suis pris le plus tôt possible, c'est à dire, dès l'instant où je savais que je m'occuperai de cette partie. Ce travail était nécessaire, car, bien que que j'en avais déjà réalisé un sur la fonction XOR, il m'était difficile, voir impossible d'en faire un sur la reconnaissance de chiffres. C'est donc pour cela que la partie la plus compliquée fut, pour moi, la recherche dans ce *monde vaste et inconnu*. Puis il m'a fallut réfléchir à une méthode pour implémenter cette fonction dans le langage C, et grâce à l'expérience acquise lors de la première partie du projet, je savais dorénavant utiliser les éléments importants, comme les *struct* par exemple. Une fois ces deux premières parties terminés, j'ai pu commencer l'implémentation de notre réseau de neurones. Cela m'a pris un peu de temps au début, il me fallait imaginer comment gérer chaque étapes. Mais une fois compris, il ne me restait plus qu'à appliquer les différents algorithmes (*forwardpropagation* et *backpropagation*).

Une fois le réseau de neurones implémenté sur la base de donnée MNIST, j'ai créé des nouvelles fonctions qui pourront reconnaître n'importe quel chiffre. Malheureusement, le programme ne marchait pas sur les chiffres du sudoku découpés... et après un travail acharné à essayer de régler ce problème, je n'ai quand même pas réussi à le faire marcher, par faute de temps probablement.

Malgré ce petit souci, je suis tout de même assez fier de ce que j'ai pu faire et je n'en repars que plus motivé à réussir la prochaine fois !

10 Conclusion

Pour conclure, nous avons pu grandement en apprendre sur le développement d'un OCR. Même si nous n'avons pas réussi à répondre complètement aux besoins de chaque partie, nous avons fait de notre mieux pour rendre un projet complet. On espère que vous serez satisfait de notre solveur de sudoku.

L'équipe KAZUKAKI.