



EPITA  
2ÈME ANNÉE CYCLE PRÉPARATOIRE

---

OCR

---

Rapport de Soutenance 1

*Auteurs*

Raphaël NAHOUM      Mathis FRIGANT  
Othmane ZIYAD      Malo BEAUCHAMPS  
Mickaël RAZZOUK

Septembre 2022 - Décembre 2022

# Table des matières

<b>1</b>	<b>Introduction</b>	<b>3</b>
1.1	Présentation du groupe . . . . .	3
1.1.1	Malo . . . . .	3
1.1.2	Micka . . . . .	3
1.1.3	Othmane . . . . .	3
1.1.4	Raphaël . . . . .	4
1.1.5	Mathis . . . . .	4
<b>2</b>	<b>Répartition des tâches et avancement</b>	<b>5</b>
2.1	Répartition . . . . .	5
2.2	Avancement . . . . .	5
<b>3</b>	<b>Notre avancement</b>	<b>6</b>
3.1	Malo . . . . .	6
3.1.1	Nuance de gris . . . . .	6
3.1.2	Binarisation . . . . .	7
3.2	Mickael . . . . .	8
3.2.1	Réseau de neurone . . . . .	8
3.2.2	Back propagation . . . . .	8
3.2.3	Fonction d'activation . . . . .	8
3.2.4	Sauvegarde d'entraînement et chargement . . . . .	9
3.2.5	Résultats . . . . .	9
3.3	Othmane . . . . .	9
3.3.1	Sudoku solver . . . . .	9
3.4	Raphaël . . . . .	10
3.4.1	I/O . . . . .	10
3.4.2	Interface utilisateur . . . . .	11

---

3.5	Mathis . . . . .	12
3.5.1	Rotation de l'image . . . . .	12
3.5.2	Détection des lignes . . . . .	12
3.5.3	Rotation automatique . . . . .	13
3.5.4	Différenciation des lignes horizontales et verticales . . . . .	13
3.5.5	Identification des cases . . . . .	14
4	Conclusion	16

# Introduction

## 1.1 Présentation du groupe

### 1.1.1 Malo

Je n'avais jamais fait de C avant l'EPITA, et ce projet m'a encore plus rassuré, j'aime beaucoup le C. Étant un fervent adorateur de Linux (I use arch BTW), et de l'open source, le C est un langage bas niveau très important. Néanmoins, je n'avais jamais fait de C avant l'EPITA, et ce projet m'a encore plus rassuré, j'aime beaucoup le C. C'est un langage très puissant quand bien utilisé.

### 1.1.2 Micka

J'ai toujours touché à l'informatique en me lançant des projets personnels, je suis très intéressé par les défis techniques à relever durant ce projet, je pense qu'il me permettra de développer des nouvelles compétences, car il aborde des points plus techniques que ce à quoi je suis habitué.

### 1.1.3 Othmane

Après une année relativement mouvementée à l'EPITA, je me retrouve aujourd'hui prêt à entamer ce projet de S3. Je m'appelle Othmane Ziyad, et j'ai dès à présent comme tâche de travailler sur ce projet. Trêve de blabla, je suis passionné par l'informatique et la programmation, mais sans grande surprise (sinon je ne serais pas à cette école). Après avoir découvert le plaisir de l'algorithmique en première avec la spécialité NSI (que j'ai bien évidemment gardée en terminale), j'ai eu la chance de développer mes compétences et prendre goût à la matière. De plus, une année d'étude à l'EPITA m'a aidé à progresser et mieux coder, ainsi que de mieux travailler en groupe. Il n'en est pas moins que j'appréhendais un peu le

projet, notamment à cause du fait que je ne j'ai jamais touché de C avant cette année. Coder une Intelligence Artificielle de reconnaissance d'image en C me semblait hors de question. Tout de même, je me suis familiarisé avec le langage et ses techniques, et j'ai fait de mon mieux pour surmonter les difficultés que j'ai rencontré. Je sais faire preuve d'une grande autonomie et adaptation. J'ai de grandes attentes pour ce projet, car non seulement est-ce pour moi un but à atteindre et un challenge à surmonter, mais il me permettra également d'acquérir des compétences que je n'avais pas auparavant, ce qui m'aidera grandement dans le futur. Je suis convaincu qu'à l'aide de mon groupe, on réussira tous à se pousser de l'avant et créer un projet digne de ce nom.

#### 1.1.4 Raphaël

Passionné d'informatique également, j'aime créer des projets avec des amis. J'aime également la rétro ingénierie (reverse engineering), j'ai principalement commencé avec du bytecode java depuis plusieurs années. Ceci m'a permis de comprendre beaucoup de concepts liés au développement de logiciels et au fonctionnement de certains systèmes informatiques.

#### 1.1.5 Mathis

Je suis passionné de cybersécurité et fais partie de l'équipe HDFR. Je m'intéresse beaucoup à l'informatique et maintenant à tout ce qui est infrastructure réseau. J'ai aussi des bases en langage bas niveau : assembleur x86 et ++C. Membre de la communauté open-source depuis quelque temps, j'apporte une grande importance au respect de la vie privée.

# Répartition des tâches et avancement

## 2.1 Répartition

Tâches	Malo Beauchamps	Mickael RAZZOUK	Othmane Ziyad	Raphael Nahoum	Mathis Frigant
Réseau de neurones		✓✓			✓
Traitement d'image	✓✓	✓		✓	
Sudoku solver	✓		✓✓		
Séparation d'image			✓		✓✓
I/O				✓✓	
UI				✓✓	

Légende :

✓✓ = Responsable de la tâche

✓ = Travaille sur la tâche

## 2.2 Avancement

Tâches	Avancement
Réseau de neurones	20%
Traitement d'image	60%
Sudoku solver	100%
Séparation d'image	60%
I/O	90%
UI	5%

# Notre avancement

## 3.1 Malo

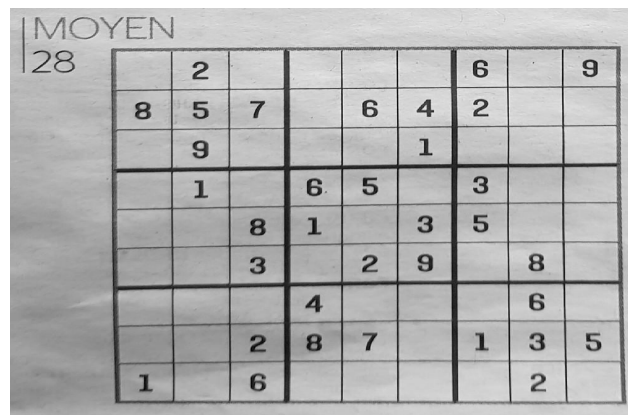
### 3.1.1 Nuance de gris

J'avais commencé ma partie avant de faire le TP SDL2 en programmation, mais ce TP m'a tout de même permis d'améliorer un peu mon code. C'était une partie qui me semblait intéressante de son point de vue mathématique. En effet, dans le but d'avoir l'image la plus nette, et la plus facilement utilisable pour reconnaître les chiffres, il faut d'abord passer par une suite d'opérations que l'image va subir.

La première étape est donc de passer d'une image en couleur, en nuance de gris. Pour cela nous avons du calculer, pour chaque pixel, la couleur moyenne, selon ses valeurs de rouge, vert et bleu. La formule utilisée est :

$$\text{average} = 0.2989 * \text{rouge} + 0.587 * \text{vert} + 0.114 * \text{bleu}$$

Voila le résultat obtenu :



MOYEN 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	5
1		6					2	

### 3.1.2 Binarisation

Une deuxième partie de ce traitement est de passer de la nuance de gris à une image en noir et blanc. Pour cela de nombreux algorithmes existent, nous avons choisis celui d'Otsu, l'Otsu Thresholding. Cet algorithme crée un histogramme des valeurs de chaque pixel, il va ensuite effectuer un calcul pour trouver le seuil (threshold), compris entre 0 et 255, ce qui va permettre de ensuite de binariser l'image. Pour chaque pixel, si la valeur du pixel est plus grande que celle du seuil, alors ce pixel sera noir, sinon, il sera blanc. On se retrouve donc avec une image contenant seulement du noir ou du blanc.

Néanmoins, il reste encore un axe d'amélioration, notamment la réduction de bruit, où la technique du flou Gaussien sera utilisée. Si encore une fois, les résultats sur certains cas précis viendraient à être peu concluant, nous pourrions toujours utiliser la technique de dilation et d'érosion, ou encore la méthode de Sauvola à la place de Otsu pour certaines applications. Voici donc le résultat final après application de la binarisation.

MOYEN  
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	5
1		6					2	

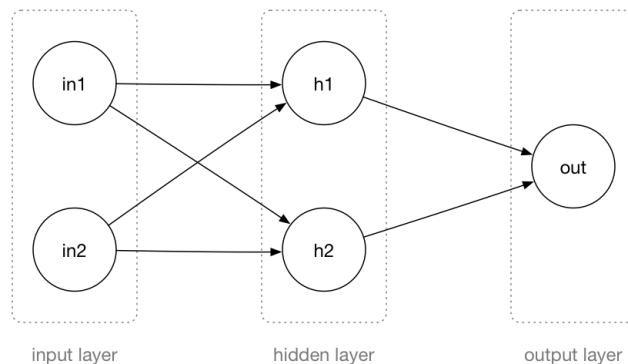


## 3.2 Mickael

### 3.2.1 Réseau de neurone

Pour cette première soutenance, le réseau de neurone a pour objectif de répliquer la fonction XOR et bien que la solution technique qu'est un réseau de neurone n'est pas adapté à ce problème, c'est un premier exercice simple qui m'a permis de plonger une première fois dans le monde de l'IA. Dans l'optique où le but final est un réseau de neurone de reconnaissance de caractère, durant l'implémentation du XOR, j'ai gardé à l'esprit la flexibilité du code pour faciliter la prochaine étape.

Voici le schéma du réseau de neurone retenu :



Deux neurones en entrée, car l'opérateur XOR est binaire, un neurone en sortie pour le résultat qui est compris entre 1 et 0. Le nombre de neurones caché lui peut être modifié, mais la configuration minimale à 2 neurones fonctionne très bien.

### 3.2.2 Back propagation

L'implémentation de cet algorithme est la partie qui implique le plus de calcul. Son rôle est de modifier le poids et biais de chaque neurone en partant du résultat dans la couche de sortie et en le comparant au résultat attendu. Au fil des epochs les corrections apportées par la back propagation font tendre les résultats vers les réponses attendues.

### 3.2.3 Fonction d'activation

Ces fonctions servent à briser la linéarité du réseau et éviter qu'il ne se bloque dans un creux. Dans un cas basique comme le XOR, je n'ai pas noté de grandes différences de performances entre les différentes fonctions.

### 3.2.4 Sauvegarde d'entraînement et chargement

Après un certain nombre d'époch d'entraînement, le réseau obtient des résultats satisfaisants, il faut donc sauvegarder le modèle de ce réseau qui est défini par les poids et biais de chaque neurone. Le modèle est sauvegardé dans un fichier texte qui peut être lu pour charger ce même modèle et ainsi retrouver ses performances sans avoir à entraîner le réseau une nouvelle fois.

### 3.2.5 Résultats

Avec les paramètres suivants :

```
inputNb 2 // for XOR we only have 2 input for 2 bits
hiddenNodesNb 2 // minimum configuration for XOR
outputNb 1 // for XOR we only have 1 bit of output
trainingSetsNb 4
learningRate 0.1
```

et 1000000 epochs. Nous obtenons de bons résultats :

```
I think the result of 0 XOR 0 is: 0.00392949
I think the result of 0 XOR 1 is: 0.996699
I think the result of 1 XOR 0 is: 0.996698
I think the result of 1 XOR 1 is: 0.00335651
```

## 3.3 Othmane

### 3.3.1 Sudoku solver

En ce qui me concerne, j'ai eu à me charger de la partie résolution du sudoku. En premier lieu, j'ai traité le problème comme si je résolvais une array d'array. Cette approche était plus simple et intuitive à mes yeux. Malgré le fait qu'on implémentera notre sudoku sous forme d'array simple, un `int[]`, j'ai préféré le traiter comme tel, et ensuite adapter mon code. L'algorithme de résolution est relativement simple, je me suis basé sur l'algorithme naïf. J'ai

donc écrit plusieurs fonctions différentes : la première analyse une grille et détermine si celle-ci est solvable ou non. Cette fonction sera réutilisée ultérieurement, notamment dans le solver, où on verra si la grille générée peut être résolue, et revenir en arrière si elle ne peut pas. La deuxième fonction détermine simplement si la grille est résolue. Cette fonction sera utile pour reconnaître le cas d'arrêt. Le solver s'arrêtera lorsque le sudoku est résolu. Finalement, le solver, qui résout le sudoku.

Le principe est le suivant : tant que la grille n'est pas résolue, l'algorithme va la modifier statiquement. Il va essayer successivement différentes combinaisons de chiffres possibles, de 1 à 9. Dès que la grille ne peut pas être résolue (ou fausse), l'algorithme revient à la dernière itération où le chiffre a été ajouté. Il essaye ensuite avec un chiffre différent. Cette fonction s'appelle donc récursivement jusqu'à résoudre la grille. Je n'ai pas traité le cas où la grille initiale ne peut pas être résolue, car je ne l'ai pas jugé pertinent. En effet, notre solver analysera une grille prise en photo pour la résoudre. Il n'est donc pas nécessaire, en théorie, de vérifier si celle-ci a une solution.

Finalement, j'ai modifié mon code pour qu'il traite des arrays plutôt que des arrays d'array. Même si cette étape peut sembler superflue et inutile, il aurait été en effet plus rapide de traiter une array dès le début, elle m'a permis de mieux visualiser le code et de perfectionner mon algorithme. J'ai donc dû revoir tout mon code depuis le début, et le modifier pour qu'il soit adapté à nos critères. Cette partie s'est montrée particulièrement difficile. En effet, il aurait été facile de remplacer tous les index  $[i][j]$  par  $[9i+j]$  car un sudoku aura toujours 9 lignes, mais le code ne marchait pas suite à cette modification. En réalité, il a fallu revoir tout le code. Le principe restait le même, mais le repère d'index a changé. Ultimement, j'ai également dû changer la nature des arrays. Je traitais originellement un char array, mais j'ai dû le modifier en int array, ce qui a simplifié les calculs d'index.

Après un travail acharné, mon solver marchait finalement. Il prenait une liste d'entiers correspondant à un sudoku non résolu, et le résout statiquement.

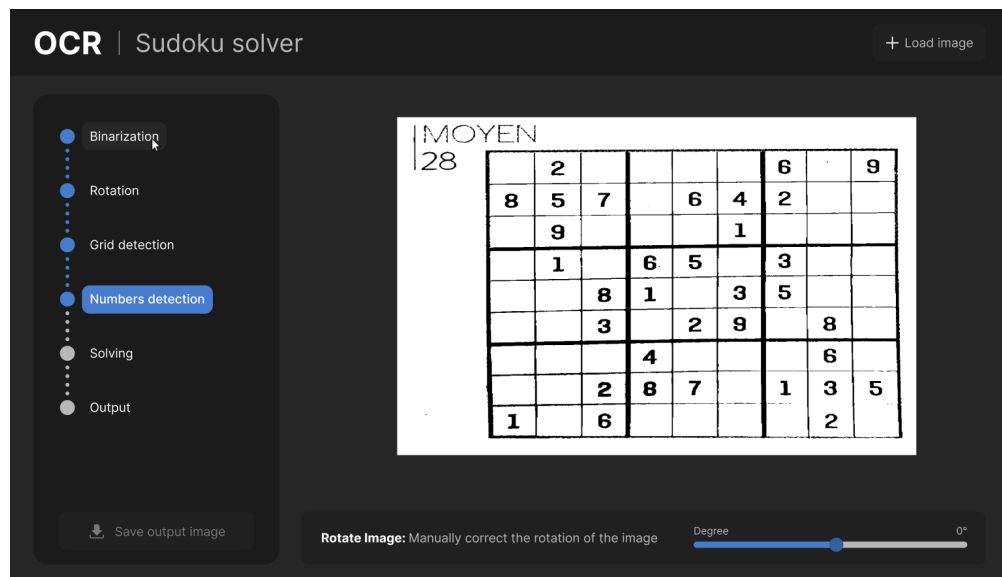
## 3.4 Raphaël

### 3.4.1 I/O

La première étape pour résoudre le sudoku est de charger un fichier afin de le donner au solver. Ces fichiers supportent plusieurs types de sudoku, suivant la taille de ceux-ci (9x9, 16x16, etc.). Cette partie consistait uniquement à réaliser un parsing simple de fichier.

### 3.4.2 Interface utilisateur

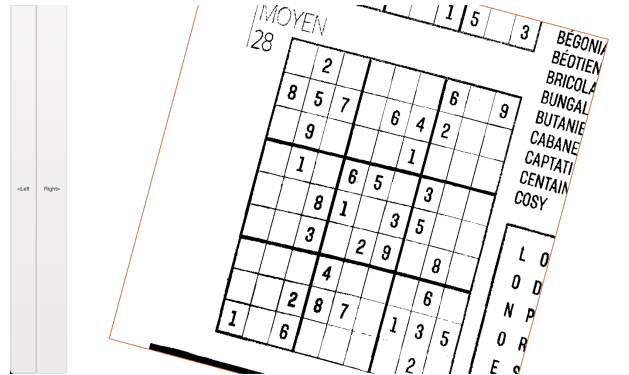
Afin d'utiliser l'OCR, il est nécessaire d'avoir une interface. Cette interface se doit d'être assez simple, intuitive, et bien évidemment agréable à regarder. C'est donc ce que j'ai essayé de faire en fabriquant cette maquette que je réaliserai par la suite avec SDL, voire GTK. Tout ceci a été conçu sur Figma, un outil très puissant permettant de concevoir des maquettes et autres. Les étapes de l'OCR sont ainsi indiquées sur la partie gauche de l'écran ou nous pouvons avoir un aperçu d'où en est le programme. Par la suite, on pourra même revenir sur chaque étape afin de visualiser l'état de l'image à cet instant. Il y aura aussi la possibilité de changer quelques paramètres manuellement, tel que la rotation d'image dans le cas où la rotation automatique ne suffirait pas. L'image produite pourra être sauvegardée à la fin du traitement afin de garder le résultat sous forme de fichier image, et non uniquement textuel tel que le programme de résolution renvoie le résultat. L'interface n'est au final qu'un lien entre chaque partie du projet, elle utilise les fonctions que chacun a créées et les réunit graphiquement pour l'utilisateur.



## 3.5 Mathis

### 3.5.1 Rotation de l'image

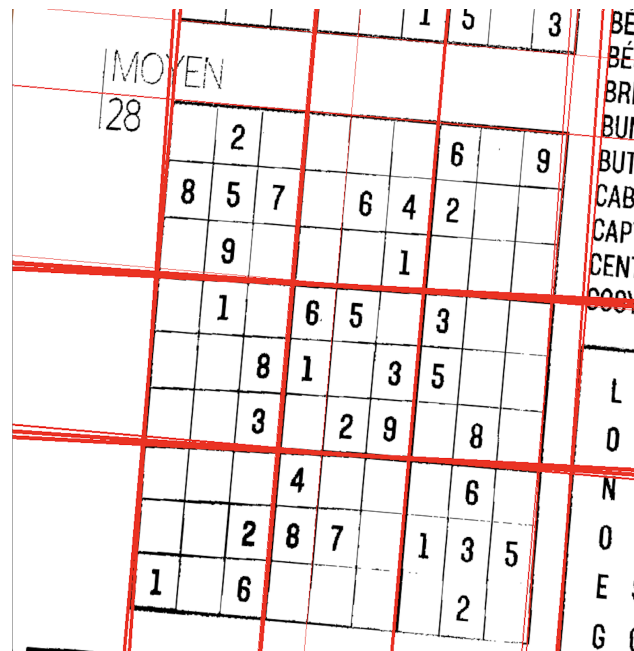
Création d'une gui simpliste avec deux boutons pour faire tourner l'image à gauche ou à droite :



J'ai recodé une fonction pour effectuer la rotation de l'image.

### 3.5.2 Détection des lignes

Pour détecter les lignes, j'ai utilisé la méthode du transformé de Hough : on exprime les coordonnées d'un point en coordonnées sphériques. on peut ensuite déterminer toutes les lignes qui passent par ce point. À l'aide d'un histogramme on peut déterminer quelles lignes sont les plus présentes et donc celle qui devraient représenter la grille. Ma première implémentation de la détection de lignes ne fut pas concluante. Les lignes détectées n'étaient pas tournées dans le bon sens. Une fois ce problème résolu, j'ai pu afficher les lignes verticales mais j'avais toujours un souci sur les lignes horizontales. J'ai réussi à régler le souci et ai pu afficher correctement les lignes sur l'image :

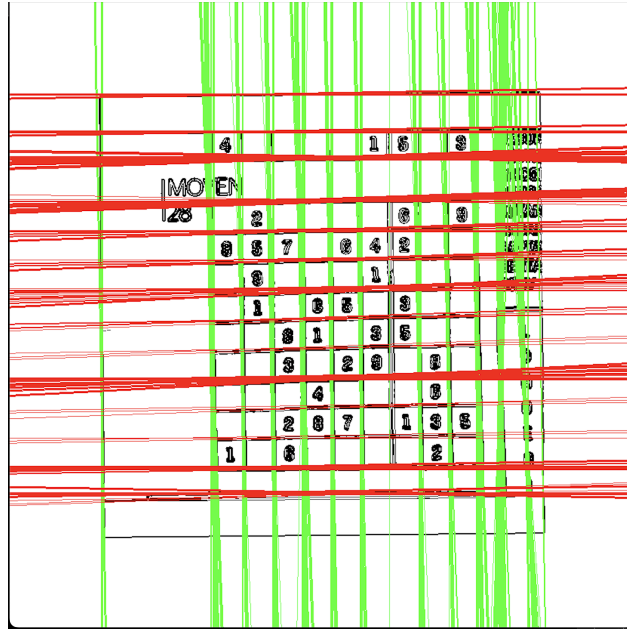


### 3.5.3 Rotation automatique

Une fois les lignes détectées, j'ai pu calculer le delta moyen de différence d'angle avec l'angle  $90^\circ$ . J'ai ainsi pu effectuer une rotation sur l'image de l'angle trouvé et ainsi la remettre droite. J'ai d'abord voulu effectuer une rotation sur les lignes que j'avais trouvé avec la première détection de lignes, mais le résultat n'était vraiment pas concluant. Pour pallier ceci, j'effectue deux passes de détection de lignes sur l'image : une première pour détecter l'orientation, puis une seconde pour détecter, pour de vrai, les lignes sur l'image.

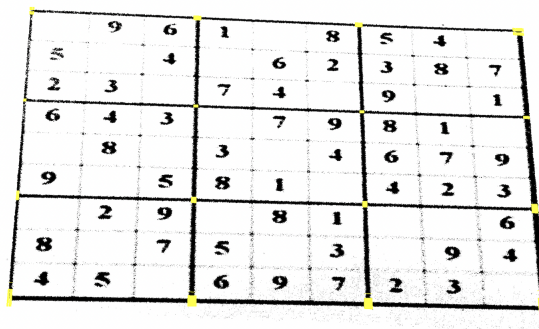
### 3.5.4 Différenciation des lignes horizontales et verticales

Une fois l'image redressée, j'ai pu isoler les lignes et les trier en fonction de si elles étaient horizontales ou verticales. Ceci va me permettre de déterminer les points d'intersection entre ces lignes pour isoler les différentes cases.

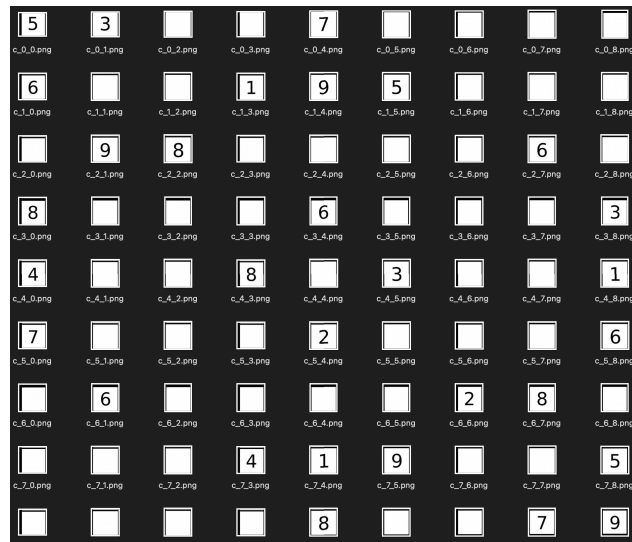


### 3.5.5 Identification des cases

Pour identifier les cases et les séparer, je détermine les points d'intersection entre les lignes verticales et horizontales, puis je calcule la distance moyenne en x et en y de ces points pour éliminer les points en trop et ne garder que les essentiels.



Je récupère ensuite chaque zone de pixels et l'enregistre





# Conclusion

En conclusion, nous sommes bien avancé sur le projet et avons respecté les échéances que nous nous étions fixées. Nous sommes confiant par rapport à la suite de ce projet et avons déjà nos idées de fixées sur ce qu'il nous reste à faire, ainsi qu'un début d'idée de qui s'occupera de quoi.