



**SIGMA VISION**

Eloi COUDERC Paul DUFOUR Malo GARNIER Josef TOURRAND



# 1 Introduction

## 1.1 Présentation du projet

Ceci est le rapport de la dernière soutenance de notre projet de troisième semestre, la résolution de sudoku présenté à travers une image. Il faut donc réaliser un OCR (*Optical Character Recognition*) dont le but est de reconnaître les chiffres que composent un Sudoku de 9 par 9 cases, ainsi que leur position sur l'image.

Ce programme devra ensuite le résoudre, puis afficher et enregistrer le Sudoku résolu dans une nouvelle image.

Voilà pour ce qui est de la présentation générale de ce que notre programme doit faire.

## 1.2 Ce rapport

En ce qui concerne ce rapport, vous pourrez trouver une table des matière regroupant les différents points abordés dans ce rapport. Ce rapport est aussi l'occasion pour nous de vous partager notre satisfaction par rapport au projet ainsi que les objectifs qui ont été accomplis durant ce court semestre. Nous finissons cette introduction en vous souhaitant une excellente et heureuse lecture !



## Table des matières

|          |   |           |
|----------|---|-----------|
| <b>1</b> | <b>Introduction</b>                             | <b>1</b>  |
| 1.1      | Présentation du projet . . . . .                | 1         |
| 1.2      | Ce rapport . . . . .                            | 1         |
| <b>2</b> | <b>L'équipe</b>                                 | <b>4</b>  |
| 2.1      | L'équipe et la répartition des tâches . . . . . | 4         |
| 2.2      | L'organisation . . . . .                        | 4         |
| <b>3</b> | <b>Réseau de neurones</b>                       | <b>5</b>  |
| 3.1      | Généralités . . . . .                           | 5         |
| 3.2      | Fonctionnement . . . . .                        | 5         |
| 3.3      | Implémentation techniques . . . . .             | 5         |
| 3.4      | Configuration . . . . .                         | 5         |
| 3.5      | Sauvegarde des poids . . . . .                  | 6         |
| 3.5.1    | SUMMARY . . . . .                               | 6         |
| 3.5.2    | DATA . . . . .                                  | 6         |
| 3.5.3    | Exemple . . . . .                               | 6         |
| 3.6      | Perspectives attendues . . . . .                | 6         |
| <b>4</b> | <b>Set d'image d'entraînement</b>               | <b>8</b>  |
| <b>5</b> | <b>Traitement d'image</b>                       | <b>9</b>  |
| 5.1      | Grayscale . . . . .                             | 9         |
| 5.2      | Binarisation . . . . .                          | 10        |
| 5.3      | Flou Gaussien . . . . .                         | 11        |
| 5.4      | Accentuation le contraste . . . . .             | 14        |
| 5.4.1    | La transformation de Sobel . . . . .            | 14        |
| 5.4.2    | Perte d'informations . . . . .                  | 15        |
| 5.5      | Opérations morphologiques . . . . .             | 15        |
| 5.6      | Détection de la grille . . . . .                | 17        |
| 5.7      | Correction de la rotation . . . . .             | 20        |
| 5.8      | Redressage de l'image . . . . .                 | 21        |
| 5.9      | Découpage de l'image . . . . .                  | 23        |
| 5.9.1    | Découpage en cases . . . . .                    | 23        |
| 5.9.2    | Recadrage du chiffre . . . . .                  | 23        |
| 5.9.3    | Redimension de la case . . . . .                | 23        |
| <b>6</b> | <b>Interface Utilisateur</b>                    | <b>24</b> |
| 6.1      | Premiere Interface . . . . .                    | 24        |
| 6.2      | Seconde Interface . . . . .                     | 25        |
| 6.2.1    | Menu d'accueil . . . . .                        | 26        |
| 6.2.2    | Menu Principal . . . . .                        | 26        |
| 6.2.3    | Menu Réseau Neuronal . . . . .                  | 27        |
| 6.2.4    | Menu Paramètre . . . . .                        | 28        |
| 6.2.5    | Menu Solved . . . . .                           | 28        |
| 6.2.6    | Le dark Mode . . . . .                          | 29        |



---

|   |           |
|---|-----------|
| <b>7 Sudoku Solver</b>                          | <b>30</b> |
| 7.1 Input . . . . .                             | 30        |
| 7.2 Output . . . . .                            | 30        |
| 7.3 Comment ca marche ? . . . . .               | 30        |
| <b>8 Reconstruction de l'image</b>              | <b>32</b> |
| 8.1 Lecture des fichiers grid . . . . .         | 32        |
| 8.2 Association des Surfaces . . . . .          | 32        |
| 8.3 Construction de la surface finale . . . . . | 32        |
| <b>9 Regrouper le projet</b>                    | <b>34</b> |
| <b>10 Management</b>                            | <b>34</b> |
| <b>11 Mirroring</b>                             | <b>34</b> |
| <b>12 Conclusion</b>                            | <b>37</b> |



## 2 L'équipe

### 2.1 L'équipe et la répartition des tâches

Notre équipe, Sigma Vision, est composé de 4 membres :

- **Malo Garnier**, qui s'occupe de tout ce qui touche au réseau de neurones (création, apprentissage, exploitation et sauvegarde des paramètres).
- **Paul Dufour**, qui s'est chargé de la détection de la grille et du solver et de rassembler le projet.
- **Josef Tourrand**, qui s'est chargé du traitement de l'image, notamment de la mise au gris et différents traitement permettant plus facilement la détection du réseau neuronal.
- **Eloi Couderc**, qui s'occupe de l'interface utilisateur et du système de génération d'image d'entraînement pour le réseau de neurones.

### 2.2 L'organisation

Dès le début du projet, un serveur Discord et une page GitHub ont été créée, pour nous permettre de plus facilement communiquer et s'organiser.

Chaque dimanche, nous nous réunissions pour présenter le travail individuel de chacun. Et sur la dernière semaine, une réunion tout les 2 jours car nous n'avions pas cours du coup nous pouvions plus se focaliser sur le projet. De plus, une page de documentation a été mise en ligne (<https://sigma-vision.github.io/doc-hugo/>).

Sur cette dernière, chaque membre du groupe peut venir apporter une explication sur ce qu'il a fait de façon qu'à la fin, la page soit une sorte de récapitulatif du travail effectué.

Github est une grande aide pour tout ce qui est gestion de fichier.

Nous avons par ailleurs mis en place un github action, qui permet de vérifier si le code compile et si le code est bien intenté. Des vérifications faciles, mais toujours utiles.



### 3 Réseau de neurones

Un des éléments clés de ce projet est la reconnaissance des caractères. Pour cela, nous avons utilisé un réseau de neurones.

#### 3.1 Généralités

Nous avons dès les prémisses de notre projet développé un système de réseau de neurone entièrement configurable et universel fonctionnant avec n'importe quel configuration.

Pour la première soutenance, le réseau avait été adapté pour la fonction XOR avec un franc succès. Pour cela, nous avons utilisé une architecture 2-3-2, qui donnait les meilleurs résultats avec une fiabilité proche des 100%.

#### 3.2 Fonctionnement

Le programme est divisé en deux parties :

- l'apprentissage
- l'utilisation

L'apprentissage est effectué en amont de la publication du projet et celui-ci est livré avec un fichier de configuration déjà paramétré. Cela permet de ne pas avoir à conserver le dataset d'entraînement dans les fichiers tout en garantissant un résultat convenant.

Néanmoins (comme Voldemort (nez en moins (un peu de rire))) l'ensemble des fichiers permettant la génération à la fois du dataset d'entraînement ainsi que de la fonction d'apprentissage sont présents dans le dépôt.

#### 3.3 Implémentation techniques

Le système de réseau repose sur l'utilisation de tableaux de pointeurs. Un réseau de neurones est structuré en plusieurs éléments :

- les entrées
- les noeuds, répartis en couches
- les biais, un par noeud
- les poids, le nombre de noeuds dans la couche précédente par noeud

Pour l'apprentissage les deltas de gradient sont enregistrés également pour la backpropagation.

Ainsi, un tableau est instantié pour chacun de ces éléments. Celui-ci contient des pointeurs (un par couche) vers d'autres tableaux qui contiennent eux-mêmes des pointeurs (un par noeud) redirigeant vers la valeur attendue. Pour les poids, un troisième niveau est nécessaire étant donné qu'il y en a plusieurs par noeud. En utilisant cette technique, la quantité exacte de mémoire nécessaire est allouée et il n'y a pas de problèmes d'accès car il n'y a pas de taille constante dans le code.

L'inconvénient de cette méthode est que le code pour parvenir aux valeurs est relativement verbeux car nous avons utilisé des noms de variables clairs donc plus longs. Toutefois, nous considérons que la lisibilité est préférable à la taille du code, sans oublier l'efficacité de celui-ci.

#### 3.4 Configuration

Configurer un réseau de neurones artificiels de manière efficace peut être un processus complexe et dépend de nombreux facteurs, tels que le type de données sur lesquelles est entraîné le réseau, le type de tâche pour laquelle le réseau est utilisé, et les objectifs que l'on souhaite atteindre.

Voici quelques étapes à suivre pour configurer un réseau de neurones artificiels de manière efficace : Choisir le type de réseau de neurones à utiliser en fonction de la tâche à accomplir. Par exemple, si l'on souhaiterait entraîner un réseau pour la reconnaissance d'images, nous aurions pu utiliser un réseau de neurones convolutionnel (CNN). Cependant cette méthode est assez complexe à implémenter, surtout



sans aucune expérience dans le domaine. Pour de la simple reconnaissance de caractères, nous avons gré le choix d'un réseau classique complet.

Il faut ensuite déterminer la structure du réseau, c'est-à-dire le nombre de couches et le nombre de neurones dans chaque couche. Vous pouvez utiliser des techniques d'optimisation comme la validation croisée pour aider à déterminer la meilleure structure de réseau.

Choisissez un algorithme d'optimisation pour entraîner le réseau. Des algorithmes populaires incluent le gradient descendant stochastique (SGD), le gradient descendant sur mini-lots (minibatch SGD) et l'algorithme Adam.

### 3.5 Sauvegarde des poids

La phase d'apprentissage génère des valeurs pour les biais et poids du réseau et il nécessaire de les sauvegarder dans un fichier afin de pouvoir utiliser le réseau sans devoir le réentraîner à chaque fois. Nous avons développé un format de fichier (`.nnconf`) pour y parvenir. Ce fichier est écrit à la fin de l'apprentissage et lu avant l'utilisation.

Le format est sauvegardé en clair et des clés sont présentes pour plus de lisibilité. Il repose sur une lecture ligne par ligne des informations avec des espaces en séparateurs. Deux sections sont présentes :

#### 3.5.1 SUMMARY

Le nombre de couches est indiqué sur la première ligne.

`1 <nb>`

Le nombre d'entrées et de noeuds est indiqué sur la deuxième ligne.

`n <nb_entrées> <nb_noeuds_couche1> [...] <nb_noeuds_sortie>`

#### 3.5.2 DATA

Pour chaque couche sont écrites deux lignes :

- Une pour les biais avec la valeur pour chaque noeud

`b <biais_noeud1> [...]`

- Une pour les poids avec les valeurs pour chaque noeud séparées par >

`w > <poids1_noeud1> <poids2_noeud1> [...] > <poids1_noeud2> <poids2_noeud2> [...]`

#### 3.5.3 Exemple

Ce fichier contient la configuration d'un réseau de deux couches, avec deux entrées, trois neurones sur la couche cachée et deux neurones en sortie. Les poids et biais des neurones sont indiqués dans la section # DATA.

```
# SUMMARY
l 2
s 2 3 2
# DATA
b -3.773078 2.522345 -9.436241
w > 8.695820 7.500989 > 3.791798 -2.807192 > 5.470344 7.158195
b 2.704783 -2.749547
w > -12.003695 3.508191 12.106470 > 12.005393 -3.461278 -12.108805
```

FIGURE 1 – Exemple d'un fichier nnconf

### 3.6 Perspectives attendues

Ayant développé une structure de réseau de neurones universelle, nous pensions que l'adaptation à la reconnaissance de chiffres serait rapide. A l'origine, nous n'aurions juste du modifier le nombre de couches et de noeuds ainsi que le dataset d'entraînement.



Toutefois cela ne s'est absolument pas déroulé comme prévu. L'application a été très compliquée et la multitude de paramètres liée aux contraintes matérielles a rendu cette tache fastidieuse.

Afin d'améliorer la précision du réseau, nous comptions utiliser une meilleure fonction d'activation. L'actuelle était une sigmoïde mais nous pensions la remplacer par une Unité exponentielle linéaire par exemple.

Cependant, ce choix n'a pas été appliqué étant donné que pour de la détection dans un spectre aussi limité la fonction sigmoïde est suffisante. En effet, l'implémentation d'une fonction RELU, GELU, SELU, etc ... entraîne la nécessité d'un système bien plus complexe afin d'éviter des dommages divers comme l'explosion de gradient, le décalage des valeurs exponentielles, le bornage de fonctions de perte,

...

C'est pour cela que nous sommes resté sur les méthodes d'origine, certes simples et élémentaires mais fonctionnelles.



## 4 Set d'image d'entraînement

Pour cette soutenance, nous étions demandé de fournir un dataset d'image de nombres qui pourront servir d'image d'entraînement pour le réseau neuronal. Lors de la recherche de ce dernier sur le net, nous sommes arrivé à la conclusion qu'il serait préférable de coder nous même une générateur de .png. Pour cela, nous avons écrit un petit code sur python qui génère un nombre prédéfini dans le code d'image dans un dossier puis ajoute chacune de ses dernières dans une archive zip, pour le gain de place.

Pour cela, nous avons utilisé différentes bibliothèques, tel que PIL (pour la création d'image) et zipfile (pour la création de l'archive zip).

```

1  from PIL import Image, ImageDraw, ImageFont
2  from zipfile import ZipFile
3  import random, os, shutil
4
5  NUMBER_OF_IMAGE = 150
6
7  #Check if the folder results exist
8  if os.path.exists("results"):
9  |   shutil.rmtree("results", ignore_errors=False, onerror=None)
10
11 os.mkdir("results")
12
13 #Generate image with random numbers and font. Save them into the result folder
14 for i in range(NUMBER_OF_IMAGE):
15     Fpath = "font/" + random.choice(os.listdir("font\\"))
16
17     img = Image.new('RGB', (15,23), color = (255, 255, 255))
18     fnt = ImageFont.truetype(Fpath, 20)
19
20     d = ImageDraw.Draw(img)
21     d.text((random.randint(-2,6),random.randint(-4, -2)), str(random.randint(0, 9)), font=fnt, fill=(0, 0, 0))
22
23     a = random.randint(0,2)
24     if (a == 0):
25         img = img.resize((8, 8))
26     elif (a == 1):
27         img = img.resize((16, 16))
28     else:
29         img = img.resize((32, 32))
30     img.save("results/" + str(i) + ".png")
31
32 #Create a zipfile archive and put all the result in it
33 with ZipFile('numberTest.zip', 'w') as zipObj:
34     arr = os.listdir("results\\")
35     for file in arr:
36         zipObj.write("results/" + file)
37
38 #Remove the results folder afterwards
39 shutil.rmtree("results", ignore_errors=False, onerror=None)

```

Ici, la variable NUMBER\_OF\_IMAGE représente le nombre d'image que le code générera.

Dans un premier temps, le script va générer un dossier *results* où sera stockés les images le temps de les insérer dans l'archive zip.

Par la suite, il va créer une image contenant un chiffre aléatoire noir d'une police d'écriture aléatoire présent dans le dossier *font* positionné à un emplacement aléatoire sur un fond blanc. Ensuite, le code va resize l'image pour qu'elle soit carré et de taille aléatoire entre 8x8, 16x16 ou 32x32 qu'il viendra pour finir stocker dans le dossier *results*. Ces actions seront répétées n fois dépendamment de la variable NUMBER\_OF\_IMAGE.

Une fois toutes les images créées, il viendra générer une archive zip puis y insérera chacun des éléments se trouvant dans le dossier *results*. Pour finir, il viendra supprimer le dossier pour économiser de la place.

Exemple de ce que le programme génère :



3 images générées par le code



## 5 Traitement d'image

Le traitement d'image a pour but de régler tous les problèmes liés à l'image. Les problèmes imaginables peuvent être :

- **La lumière** Selon les photos, la lumière peut varier. En effet, elle dépend de l'environnement de la personne qui prends la photo, de la qualité de l'appareil photo, et de pleins de facteurs différents.
- **La déformation de perspective** La grille est très souvent un trapèze sur une photo, si l'appareil n'est pas en face de la grille. Les caractères apparaissent donc déformés.
- **La rotation** Les images fournies par l'utilisateur ne sont pas forcément droite. Il est donc nécessaire de la remettre droite.
- **Le zoom** Les chiffres peuvent ne pas être très lisibles, ou flous et donc complexifier la tâche de la reconnaissance de chiffres.
- **Les déformations** les surfaces ne sont pas forcément droites, et donc peuvent poser problème. Les grilles peuvent aussi être pliées.

Pour régler les problèmes, une série d'étape est nécessaire :

- **Binarisation**
- **Grayscale**
- **Flou Gaussien**
- **Accentuation du contraste**
- **Opérations morphologiques**
- **Détection de bords**
- **Détection de la grille**
- **Correction de la rotation**
- **Redressage de l'image**
- **Découpage de l'image**

### 5.1 Grayscale

Le grayscale a pour but de transformer l'image en nuance de gris.

Pour cela nous utilisons la recommandation 601 pour les couleurs non linéaires de la Commission Internationale de l'éclairage. En effet cette recommandation est avec la correction du gamma. Voici donc la formule : Gris = 0.299 \* Rouge + 0.587 \* Vert + 0.114 \* Bleu. Cette transformation est appliquée à chaque pixel de l'image, afin de pouvoir exprimer la couleur d'un pixel en une seule donnée au lieu de 3.

Comme on peut le voir dans cet exemple, l'application du flou gaussien avec un petit kernel permet de filtrer les informations inutiles de l'image, et ne va pas modifier la structure de l'image en général.



## 5.2 Binarisation

La binarisation est une étape essentielle du pré-traitement de l'image. En effet, la binarisation va permettre d'éliminer un maximum des variables externes telles que la lumière ou le fait que la grille de sudoku soit d'une couleur différente de la plupart. Il était donc crucial d'avoir un algorithme de binarisation qui soit performant dans tout genre de situation, tout en étant rapide.

C'est pourquoi nous avons décidé d'implémenter la **binarisation d'Otsu**, avec une normalisation de l'histogramme afin d'augmenter les performances de la binarisation.

La binarisation d'Otsu va tout d'abord dresser un histogramme de l'image. On applique ensuite une normalisation de l'histogramme sur l'image. Cette étape consiste à raccourcir l'histogramme et à le rendre plus équilibré.

On calcule ensuite la variance maximale de l'image : On va poser une valeur de seuil tel que tous les pixels ayant une valeur plus basse font partie de l'arrière-plan de l'image, et tous les pixels ayant une valeur plus haute font partie du premier plan.

On calcule ensuite le poids et le  $\mu$  du premier et de l'arrière plan. tel que avec t le seuil choisi

$$Wb = \sum i = 1^t histogram[i]$$

$$\mu b = \sum i = 1^t histogram[i] * i$$

On calcule enfin la variance avec la formule suivante :

$$\mu = \mu b - \mu f$$

$$var = Wb * Wf * \mu * \mu$$

On prend donc la variance maximum et on la choisit en tant que seuil.

La binarisation d'Otsu va permettre de déterminer une valeur seuil en dessous de laquelle tous les pixels sont considérés comme appartenant au second plan, tandis que tous les pixels ayant une valeur supérieure vont être considérés comme des pixels appartenant au premier plan. L'algorithme de la binarisation d'Otsu va chercher à maximiser une variance, et la valeur ayant la plus grande variance est celle qui sera choisie en tant que valeur seuil.

Nous avons aussi expérimenté avec la **binarisation de Niblack**, cependant les résultats étant bien inférieurs à ceux de la binarisation d'Otsu, nous finalement choisi la binarisation d'Otsu. La binarisation de Niblack est une binarisation qui prend en compte la moyenne des valeurs locales, aussi elle peut permettre d'éliminer des contraintes de lumière, mais présentait finalement bien plus d'inconvénients que d'avantages.

Comme l'on peut le voir dans les figures 2 et 3, la binarisation d'Otsu permet de neutraliser grandement l'image, et rend donc tout le reste du prétraitement de l'image bien plus simple.



### 5.3 Flou Gaussien

Le **Flou Gaussien** permet d'éliminer certaines imperfections de la grille, telles que des lignes incomplètes, qui peuvent rendre difficiles l'application d'autres algorithmes.

Pour l'utilisation du flou gaussien, nous avons implémenter un flou gaussien avec un radius de 3, afin de garder un maximum d'informations, et avons simplement appliqué une convolution de ce kernel sur l'image.

Voici le kernel qui est utilisé dans notre implémentions du flou gaussien.

$$\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

La taille de ce kernel étant petite, seules les imperfections minimes seront effacées, et les nuances entre les différents chiffres seront gardées.

Comme on peut le voir dans ces exemple4, l'application du flou gaussien avec un petit kernel permet de filtrer les informations inutiles de l'image, et ne va pas modifier la structure de l'image en général.

L'impact du flou gaussien est minimale sur l'ensemble de l'image, et ne va gener aucunement à sa compréhension par nos algorithmes, mais permet tout de même d'éliminer certaines petites imperfections en évitant d'employer les opérations morphologiques qui peuvent, elles, avoir un gros impact sur la compréhension de l'image.

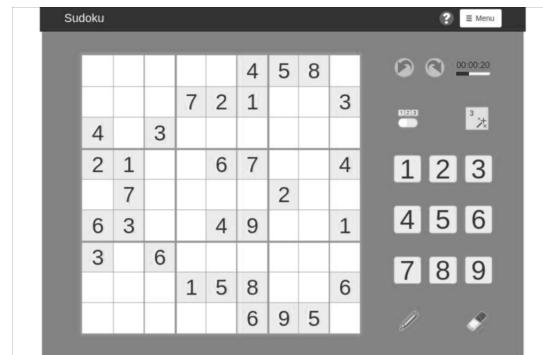


FIGURE 2 – Image avant l'application de la binarisation d'Otsu

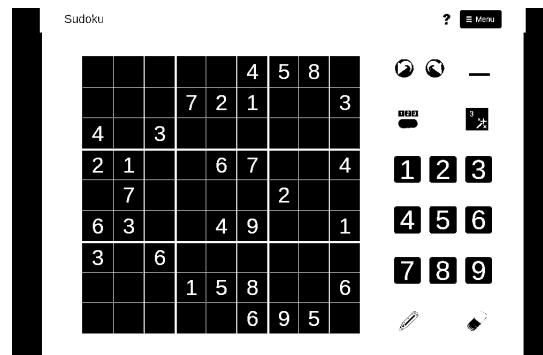


FIGURE 3 – Image après l'application de la binarisation d'Otsu

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

FIGURE 4 – Image avant l'application du Flou Gaussien

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

FIGURE 5 – Image après l'application du Flou Gaussien

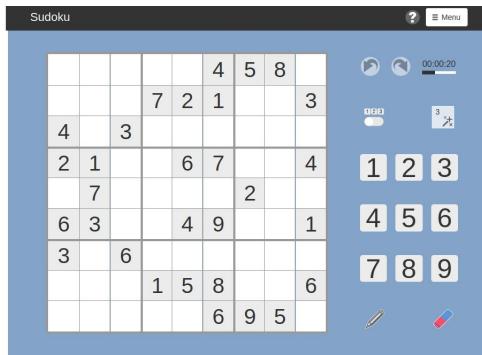


FIGURE 6 – Image avant l’application de la transformation de Sobel

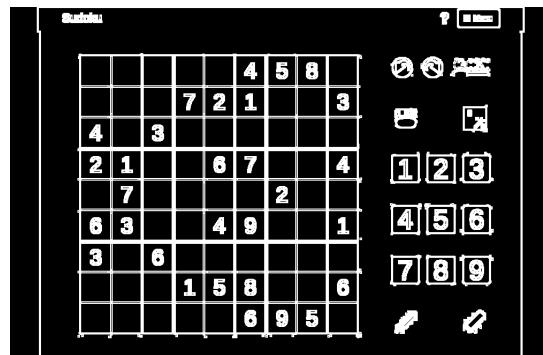


FIGURE 7 – Image après l’application de la transformation de Sobel

## 5.4 Accentuation le contraste

Le **Filtre de Sobel** est une transformation que nous avons implémenter afin d’accentuer fortement le contraste de l’image. Cela a plusieurs avantages :

### 5.4.1 La transformation de Sobel

La transformation de Sobel permet de sélectionner les informations nécessaires, en se concentrant uniquement sur les espaces avec un fort contraste et permettant ainsi de détecter la grille beaucoup plus facilement. On ignore donc de nombreux détails pouvant gêner la détection de la grille. On peut voir dans cet exemple6 que la 2nde image possède un beaucoup moins des détails que dans l’image de départ7.

La transformation de Sobel fonctionne de la façon suivante : On construit 2 kernels G<sub>x</sub> et G<sub>y</sub>, tels que G<sub>x</sub> fasse ressortir les contrastes verticaux, et G<sub>y</sub> fasse ressortir les contrastes horizontaux.

On utilise des kernels de taille 5,5, avec G<sub>x</sub> :

$$\begin{pmatrix} -5 & -4 & 0 & 4 & 5 \\ -8 & -10 & 0 & 10 & 8 \\ -10 & -20 & 0 & 20 & 10 \\ -8 & -10 & 0 & 10 & 8 \\ -5 & -4 & 0 & 4 & 5 \end{pmatrix}$$

et G<sub>y</sub> :

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

FIGURE 8 – Image après l’application de la transformation de Sobel et de la détection de la grille, avec une utilisation de la copie

$$\begin{pmatrix} -5 & -8 & -10 & -8 & -5 \\ -4 & -10 & -20 & -10 & -4 \\ 0 & 0 & 0 & 0 & 0 \\ 4 & 10 & 20 & 10 & 4 \\ 5 & 8 & 10 & 8 & 5 \end{pmatrix}$$

#### 5.4.2 Perte d’informations

Un point négatif de la transformation de Sobel est la perte d’informations qui peut s’ensuivre, comme il est constatable dans la 2nde image 6. En effet, on peut remarquer dans cette image que les chiffres sont beaucoup plus gros que la normale, et que certains sont même vides. Cela pourrait fausser les résultats obtenus par le réseau de neurones, aussi il est plus sage de ne pas ignorer cette problématique. Afin de contourner cela, nous utilisons une copie de l’image, et gardons en mémoire les 4 coins de la grille, que nous réutiliserons et modifierons dans la suite du programme. Comme il est possible de le voir dans cette image 8, la perte d’informations, surtout au niveau des chiffres, est minime.

### 5.5 Opérations morphologiques

Cependant, même si la transformation de Sobel permet d’ignorer de nombreuses données insignifiantes, elle en crée aussi de nombreuses. C’est là qu’on peut utiliser les opérations morphologiques, la dilation et l’érosion. La dilation est l’action d’appliquer un masque sur chaque pixel de l’image. Par exemple, nous appliquons le masque par défaut, qui peut être représenté par la matrice suivante :

$$\begin{pmatrix} 0 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 0 \end{pmatrix}$$

Cela signifie que tous les pixels correspondant à des 1 sur la matrice appliquée sur chaque pixel localement deviendra un pixel noir s’il n’en est pas déjà un, ou un pixel blanc dans notre cas, alors que ceux correspondant à des 0 seront ignorés.

Il va de pair avec le procédé inverse, l’érosion. L’érosion est un procédé qui peut être traduit par "transformer les bords des composantes connexes de 1 en 0". Cela signifie qu’un pixel 1 n’ayant pas ou peu de voisins va être remplacé par un pixel 0, et ce afin de réduire le bruit dans l’image. En réduisant les composantes connexes, nous avons cependant encore un peu de perte d’informations, alors nous appliquons encore une fois ce procédé sur notre copie de l’image.

On appelle l’action d’effectuer une dilation après une érosion afin de supprimer les plus petits éléments de l’image un **closing**. Ce procédé est extrêmement efficace pour détruire les liens entre plusieurs composantes connexes de pixel blanc, ce qui est crucial pour la bonne application de notre algorithme de détection de la grille, comme nous pouvons le voir ici!



FIGURE 9 – Image après une égalisation de l'histogramme et plusieurs closing

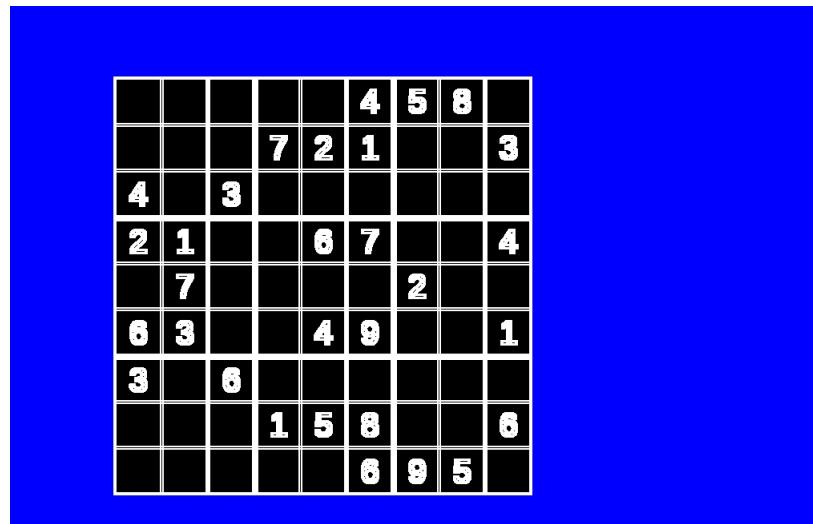


FIGURE 10 – Image apres l’application du processus pour trouver la grille

## 5.6 Détection de la grille

Pour détecter la grille nous utilisons une sorte **Labelling connected components**. Pour résumer, un label va être attribué à tout les groupes de pixels existants. Nous créons donc une grille de la même taille que nous remplissons avec ses valeurs. Ensuite il faut repasser sur la grille pour vérifier que deux labels différents ne se touchent pas.

Ensuite il faut trouver le label le plus utilisé pour ne garder que lui, et remplacer les autres par la couleur du background. Comme par magie si l’on affiche l’image du sudoku, il ne reste plus que lui ! Pour faire cette partie, il nous faut donc trouver les points des extrémités de l’image. Pour cela, nous parcourons toute la grille de labels jusqu’à trouver le label voulu en question. Nous créons ensuite une structure pour retenir les différents points.

```
typedef struct Dot
{
    int X;
    int Y;
} Dot;

typedef struct Square
{
    Dot topLeft;
    Dot topRight;
    Dot bottomLeft;
    Dot bottomRight;
} Square;
```

Cela nous permet de sauvegarder les différentes extrémités dans un **Square**.

Ensuite, on passe au calcul de l’aire du rectangle. Le rectangle ayant la plus grande aire est le sudoku. Afin de ne pas utiliser toute la mémoire du pc, nous calculons l’air que des 2 plus grandes surface. Ensuite toutes les pixels qui ne sont pas dans le label trouvé sont transformés en bleu.

A la fin nous avons donc une image propre me contenant que le sudoku.

Il nous fallait maintenant trouver les 4 coins du carré. Nous avons essayé de nombreuses techniques : - chercher les points les plus près des bords - corriger en remontant / descendant un peu - ... la technique la plus simple était au final juste de trouver le centre de l’image puis de prendre les 4 coins les plus loin du centre du carré. Avec cette technique les points trouvés dans tout les cas sont

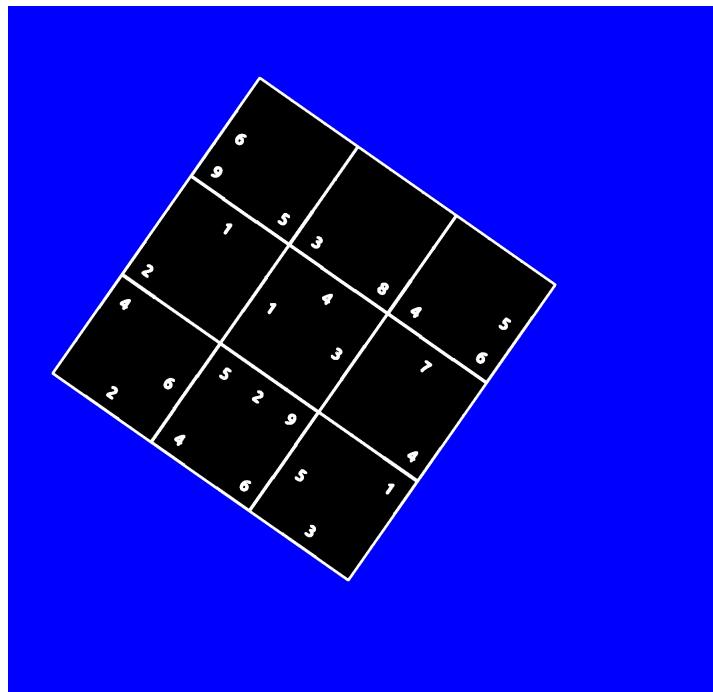


FIGURE 11 – Autre image apres l'application du processus pour trouver la grille

les bons !

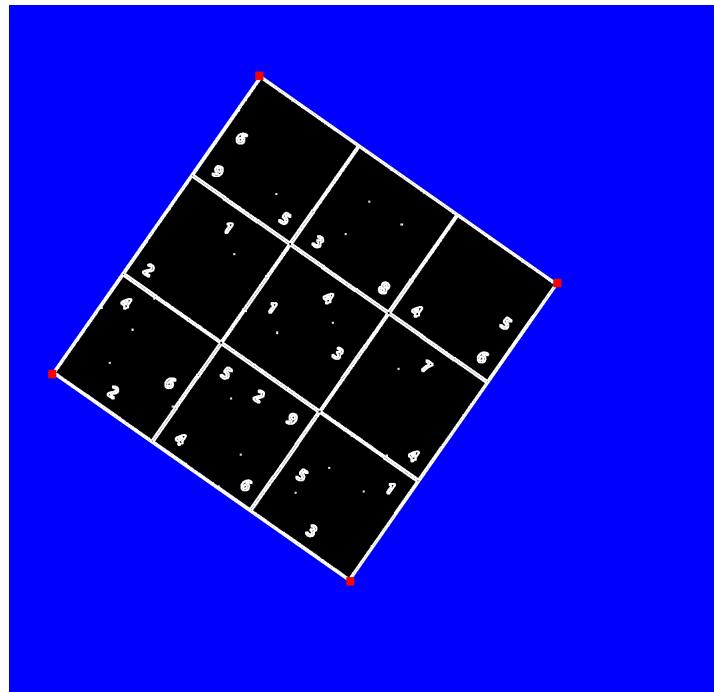


FIGURE 12 – Autre image apres l'application du processus pour trouver la grille

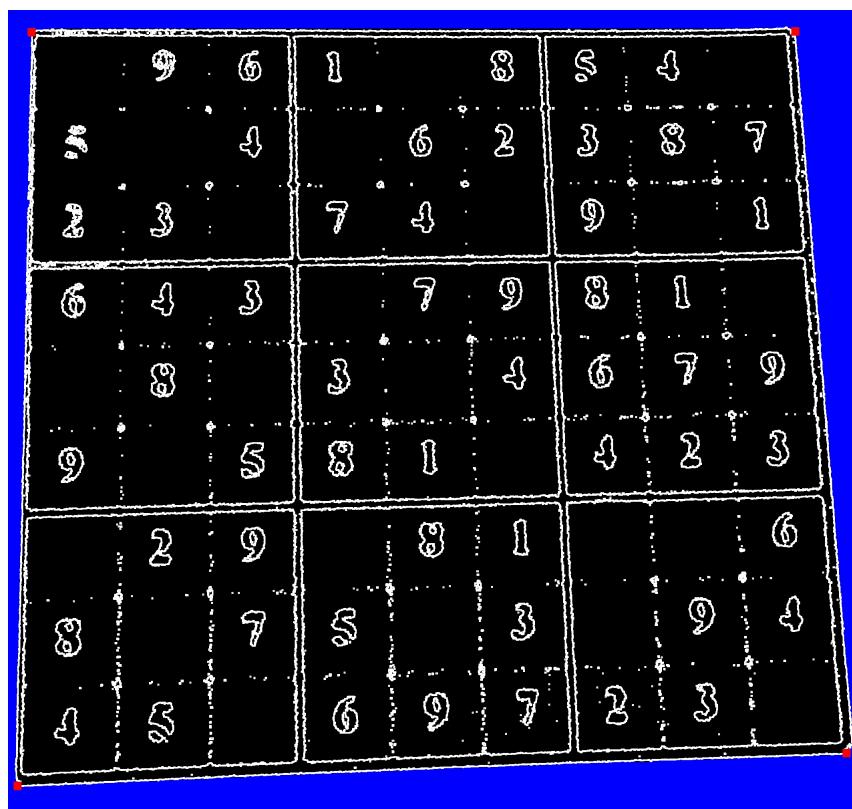


FIGURE 13 – Autre image apres l'application du processus pour trouver la grille



## 5.7 Correction de la rotation

Afin de corriger la rotation de l'image, il faut tout d'abord déterminer l'angle de la rotation à appliquer. Pour cela, l'algorithme de détection de grille va retourner les coordonnées des 2 points en haut à gauche et à droite de la grille dans une struct Square. On applique ensuite la formule suivante sur les coordonnées de ces 2 points, afin de déterminer l'angle de rotations en radians. Cette formule de trigonométrie simple va permettre de déterminer l'angle entre ces 2 points en supposant que leurs coordonnées forment un triangle rectangle avec la coordonnée souhaitée.

Soient  $(x_1, y_1)$  les coordonnées du point en haut à gauche et  $(x_2, y_2)$  les coordonnées du point en haut à droite :

$$-\arctan\left(\frac{y_2 - y_1}{x_2 - x_1}\right)$$

On applique ensuite une rotation en déterminant la position chaque pixel de la nouvelle image sur l'ancienne image : Soient  $(x, y)$  les coordonnées du pixel sur la nouvelle image,  $(xp, yp)$  les coordonnées du pixel sur l'ancienne image,  $(cx, cy)$  les coordonnées du centre de l'image et  $a$  l'angle de rotation

$$\begin{aligned} xp &= (x - cx) * \cos(a) + (y - cy) * \sin(a) + cx \\ yp &= -(x - cx) * \sin(a) + (y - cy) * \cos(a) + cy \end{aligned}$$

D'après ces coordonnées, on va ensuite pouvoir déterminer la couleur de chaque pixel de la nouvelle image **sans perte d'informations ni de pixels avec une couleur non définie**, qui peuvent arriver du aux approximations faites avec les flottants.

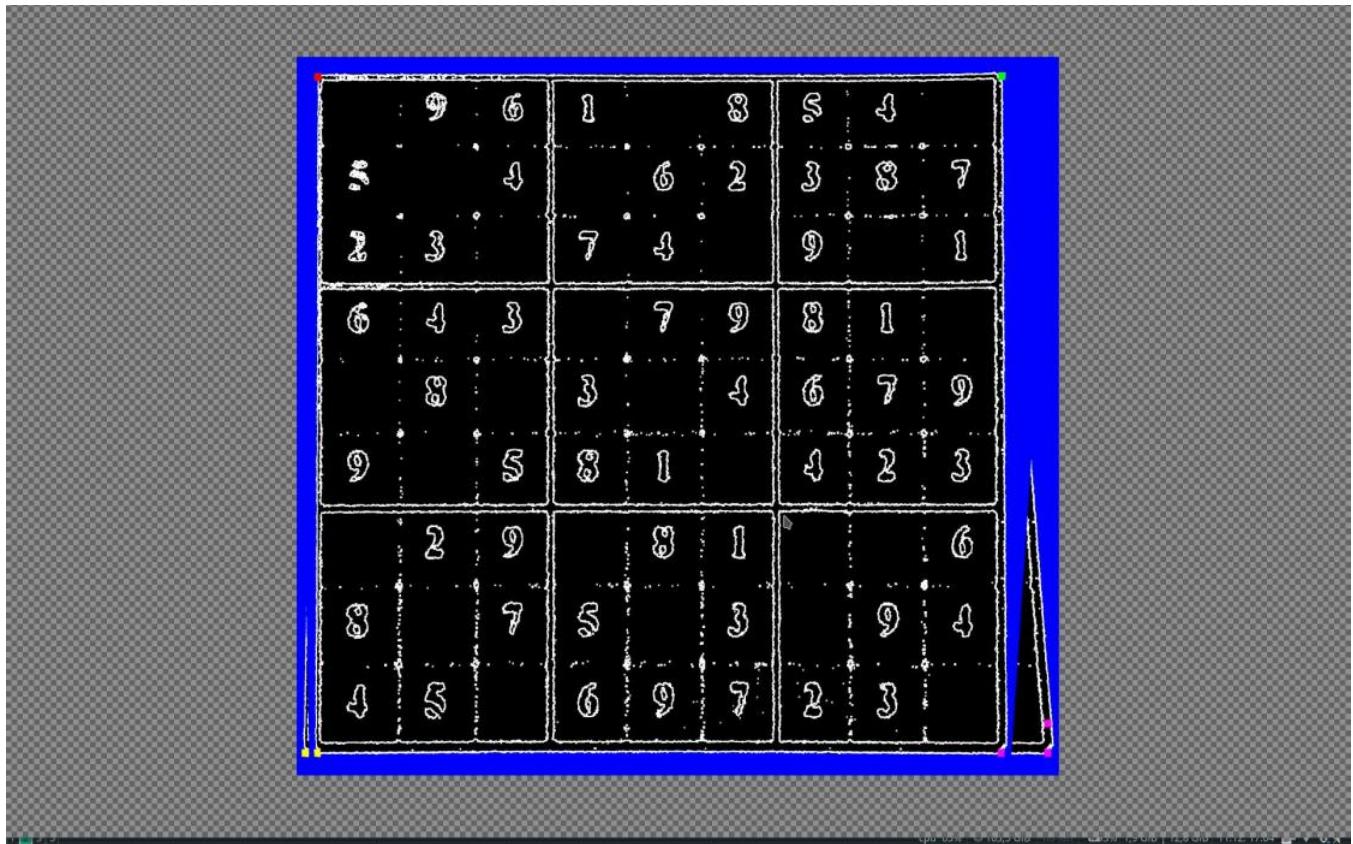


FIGURE 14 – Image qui n'était pas un carré désormais redressé

## 5.8 Redressement de l'image

Il est possible, et même assez courant que, du à une mauvaise photographie donnée en input de notre programme, la grille de Sudoku ne soit pas un carré, comme par exemple sur cette image 15.

Or, notre découpage de grille suppose, par simplicité, que c'est le cas. On voit donc apparaître la nécessité du redressement d'une image. Le redressement de l'image va s'appuyer sur le fait que la rotation automatique a pu établir une ligne droite totalement horizontale entre le point en haut à gauche de la grille, et le point en haut à droite. Nous allons donc ensuite appliquer une rotation automatique avec comme centre **chacun des coins autre que celui en haut à gauche**.

De plus, nous allons pondérer cette rotation, afin d'éviter que l'entièreté du carré ne se tourne. Cela aura comme effet d'atténuer la rotation dans les pixels les plus éloignés du centre de la rotation, et nous permettra donc d'avoir des angles de 90° à chaque coin, et donc d'avoir une grille carrée. Voici le résultat de cet algorithme : 14



FIGURE 15 – Image en input du programme qui n'est pas un carré



FIGURE 16 – Exemple de case découpée par notre programme

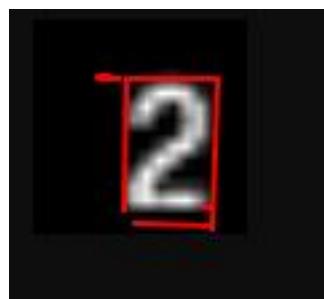


FIGURE 17 – Exemple de chiffre détecté par l'algorithme

## 5.9 Découpage de l'image

### 5.9.1 Découpage en cases

L'image doit enfin être découpé en une image par cases, afin de l'envoyer au réseau de neurone. Pour cela nous réutilisons les coordonnées du square précédent, et nous les divisons par 9 afin d'obtenir les cases. On utilise ensuite une fonction **GridCropping**, qui va permettre de recadrer une image en fonction des coordonnées d'un Square.

De plus, nous allons prendre une marge de **1/8** de la largeur et de la longueur de l'image, afin de couper distinctement les cotés de la case, qui peuvent être restés après le découpage de l'image.

Nous nous retrouvons donc avec 81 images telles que celle ci 16 :

### 5.9.2 Recadrage du chiffre

Afin d'améliorer la précision du réseau de neurones, nous avons décidé de centrer les chiffres, et ce grâce au même algorithme que nous avons utilisé pour détecter la grille. Nous détectons donc le chiffre, qui est la plus grande composante connexe de la case, et prenons ses coordonnées. Nous allons ensuite transformer l'image telle que le chiffre soit au centre de l'image, et que le chiffre ait une marge de quelques pixels. Voici un exemple de l'algorithme de détection de chiffre : 17

### 5.9.3 Redimension de la case

Ces images seront ensuite transformées telles qu'elles soient de dimensions 16x16 afin de les donner en input au réseau de neurones, sous la forme de tableau de float de taille 16\*16, une par une. Cela va donner des chiffres, qui seront écrits dans un fichier "grid" qui sera à l'endroit où le programme a été exécuté, et qui permettra ensuite de solve le sudoku !

## 6 Interface Utilisateur

La création de l'interface utilisateur a été divisé en 2 parties :

- Ce qui a été fait pour la première soutenance, qui était un moyen de prendre en main les différentes logiciels et bibliothèque utilisé pour la réalisation de l'UI
- Ce qui a été fait pour la soutenance final, qui est une refonte total de cette dernière, est qui sera notre interface final.

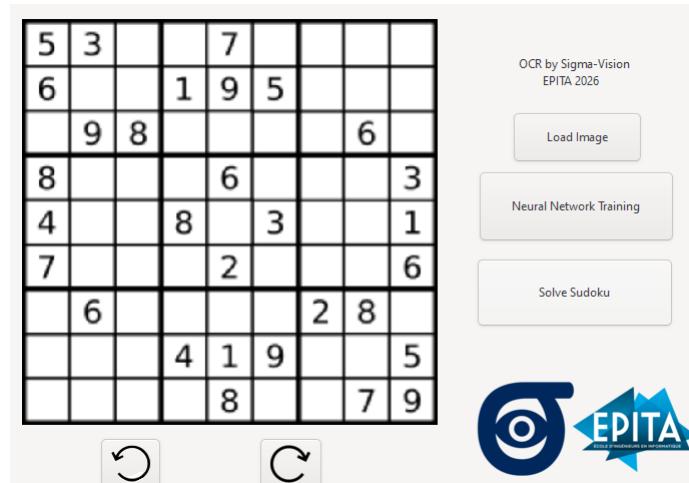
Nous allons commencer par rappeler ce qui a été fait pour la 1ere soutenance :

### 6.1 Première Interface

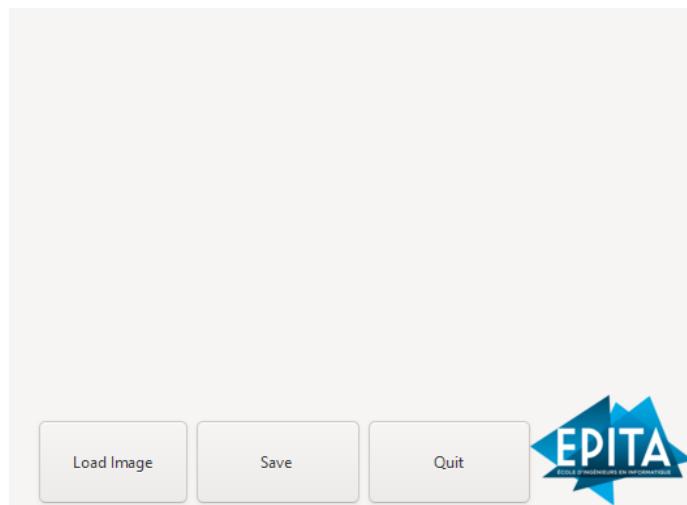
Pour la première soutenance, nous nous étions donné l'objectif de commencer une Interface Utilisateur pour prendre bien en main les différents logiciels et librairies utilisé pour sa réalisation. Pour cette dernière, nous avons utilisé la bibliothèque logiciel GTK, et le logiciel de création d'interface Glade. Ce dernier produit des fichier .glade pouvant être utilisé par GTK, générant tout le front de l'interface. Il ne reste plus qu'à coder tout le back avec GTK.

L'interface était divisée en 3 menu :

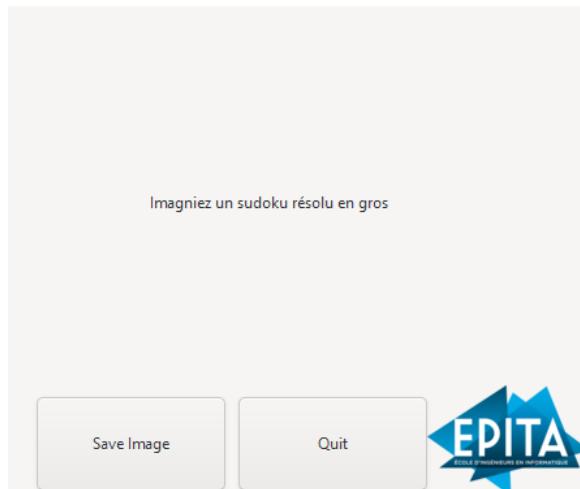
- Le menu principal, qui comportait plusieurs boutons, notamment celui du chargement de l'image, celui d'accès aux autres menu et ceux qui auraient servi de correction manuelle pour la rotation au cas où la rotation automatique ne serait pas parfaite. De plus, lorsqu'une image était chargé grâce au bouton, elle était resize pour l'affichage de façon à ce qu'elle rentre dans le carré qui lui est dédié pour s'afficher à gauche de l'interface. Les boutons de chargement d'image et d'accès aux autres menu étaient fonctionnels. Ceux de la rotation n'étaient pas encore implémenté.



- Le menu d'entraînement du réseau neuronal. Celui-ci était censé servir d'interface où l'on pouvait déposer les images que l'on souhaitait utiliser lors de l'entraînement du réseau de neurone. Il était composé d'une liste qui était censé se remplir avec les images que l'on ajoutait à l'entraînement et de plusieurs boutons en bas, notamment celui du chargement des images, celui qui sauvegardait le choix des images et celui qui permettait de quitter le menu. Ce menu n'avait encore aucune fonctionnalité sauf le bouton QUIT.



- Le menu de la grille résolue. Cette dernière était censé afficher la grille de sudoku résolue en gros et permettait la sauvegarde de cette dernière en image sous le format .png. Les deux boutons présent sur ce menu étaient tous deux fonctionnels (celui de sauvegarde de l'image ouvre une fenêtre où l'on pouvait choisir un fichier de son ordinateur ou l'on veut que l'image soit sauvegardée).



L'interface n'était pas relié au reste du projet. Elle était juste une esquisse de l'UI actuelle, une façon de bien comprendre et de s'entraîner sur les logiciels et bibliothèques. C'est pour cela que la mise en page des différents menu a été complètement retravaillé.

## 6.2 Seconde Interface

Pour la seconde interface, nous avons tout recommencé, que ce soit le fichier .glade que le fichier .c. Pour cette dernière, nous avions comme objectif de fusionner toutes les fenêtres ensemble pour que tout soit rassemblé en une seule, de fluidifier l'application en créant différents menu interchangeable au sein d'une même fenêtre et de donner une identité propre à cette dernière.



### 6.2.1 Menu d'accueil

Lorsque l'on execute le programme, la page d'accueil apparait. Cette dernière est composé de notre logo, de nos noms, de quelque png décoratif et d'un bouton "Démarrer". Ce dernier, lorsque pressé, enclanche une transition vers le Menu Principal, la ou tout le coeur de l'OCR se déroulera.



Menu d'Accueil

### 6.2.2 Menu Principal

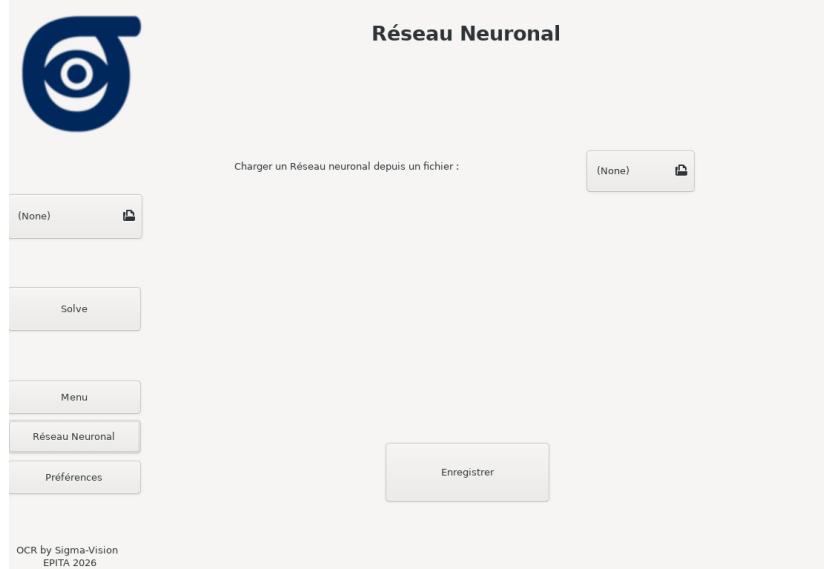
Ce menu est le menu principal. Commençons par parler de la structure global du menu. Ce menu, comme tout les autres (exempté du menu d'accueil) est composé a gauche d'un élément permettant de naviguer entre les menus. Ce menu est donc accessible en cliquant sur le bouton intitulé "Menu". A droite, au niveau du logo Sigma Vision, lorsque une image sera chargé grace au bouton en dessous du logo a gauche (ce dernier ouvre une fenetre de dialog dans laquel l'utilisateur peut naviguer au sein de ses fichiers et selectionner une image), cette dernière apparaitra a la place du logo, redmitionné de facon a ce que l'image couvre toute la surface qui lui est alloué. De plus, en bas, 2 bouttons sont disposés permettant la rotation manuelle de l'image de 90° a gauche et a droite. Lorsque l'utilisateur clique sur le bouton Solve, ce dernier lance tout le préprocess de l'image, affiche la grille traité, appelle l'IA pour qu'elle reconnaisse les digits, puis appelle la reconstruction de la grille pour qu'un png comportant le résultat soit affiché. Il était censé avoir une barre de progression en bas juste au dessus des boutons de rotations mais étant donné que le préprocess s'execute relativement rapidement, nous l'avons pas implémenté dans le back, mais seulement laissé ici pour l'estethique.



Menu Principal

### 6.2.3 Menu Réseau Neuronal

Ce menu, accessible en cliquant sur le bouton intitulé "Réseau Neuronal" est le bouton servant au chargement d'un fichier .nnconf (fichier de sauvegarde contenant l'ensemble des poids et des biais de chaque neurone) dans le réseau de neurone. Pour cela, il suffit de cliquer sur le bouton. Ce dernier, comme le bouton load image, va ouvrir une fenêtre de dialog permettant à l'utilisateur de choisir un fichier dans ses dossier puis, lorsque Enregistrer est cliqué, le fichier sera envoyé à l'IA.

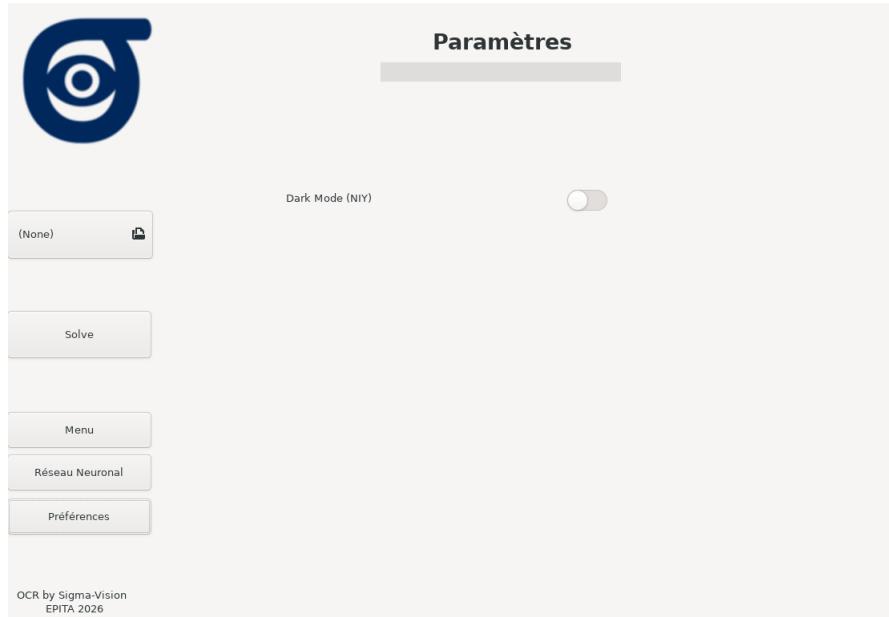


Menu Réseau Neuronal



#### 6.2.4 Menu Paramètre

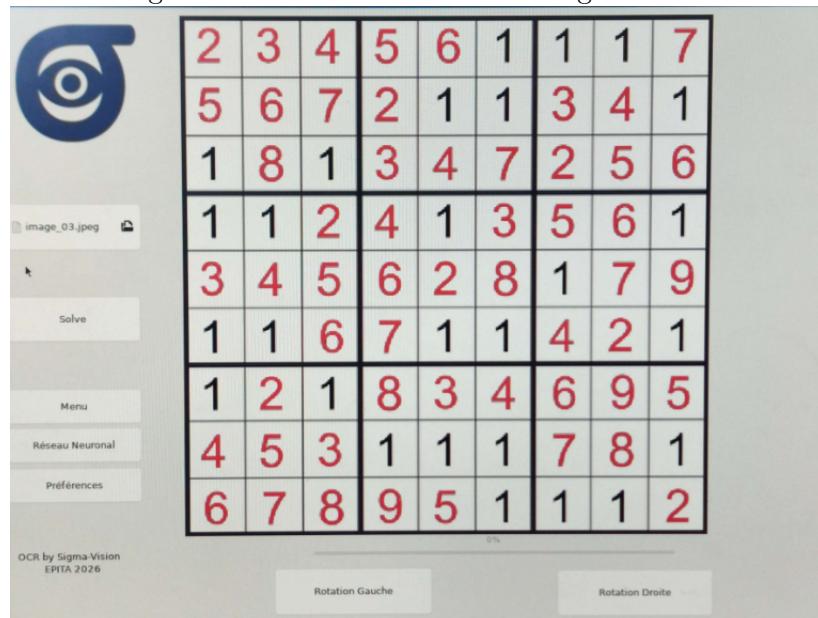
Ce menu, accessible en cliquant sur le bouton intitulé "Préférence", est seulement composé d'un switch permettant d'activer le dark mode.



Menu Paramètrel

#### 6.2.5 Menu Solved

Ce menu, apparaissant à la fin du procédé de résolution de la grille en cliquant sur Solve, est composé d'une image représentant le sudoku résolu avec en noir les chiffres déjà présents, et en rouge, les chiffres ajouté grâce au solver. En dessous de l'image, un bouton permettant de sauvegarder l'image est présent. Ce dernier, comme le bouton Load, ouvrera une fenêtre de dialog. Cette dernière permettra de choisir un dossier ou l'image résultante de l'OCR sera sauvegardée.



Menu Solvedl



### 6.2.6 Le dark Mode

Le menu étant de base blanc, comme tout bon développeur, nous avons cherché à ajouter un dark mode qui permettra d'avoir un affichage qui fait un peu moins mal aux yeux. Pour cela, nous avons créé des fichiers CSS. Ces derniers seront chargés via GTK et permettront de changer l'apparence des Widgets disposés sur l'interface (couleur des boutons, de la fenêtre et du texte). Voici quelques exemples des différentes pages du menu lorsque le dark mode est activé.





## 7 Sudoku Solver

Le solver du sodoku est une partie relativement simple, mais importante.

### 7.1 Input

En entrée ce programme prends un nom de fichier **\$name**.

Un fichier valide doit avoir :

- un . a la place des cases vides
- des espaces pour représenter les barres verticales des grilles de sudoku
- des retours a la ligne a la fin des 9 cases
- des retours a la ligne pour représenter les barres horizontal des grilles de sudoku

Par exemple ce fichier est celui donné dans la consigne.

---

```
... ... 4 58.
... 721 ... 3
4.3 ... ...
1. .67 ..4
.7. ... 2..
63. .49 ..1

3.6 ... ...
... 158 ..6
... ..6 95.
```

---

### 7.2 Output

En sortie, ce fichier sortira un fichier de nom **\$(name).result**.

Le fichier respectera les mêmes spécifications. Le résultat du fichier précédent donnera

---

```
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
```

---

### 7.3 Comment ca marche ?

Ce programme marche depuis un simple **backtracking algorithm**.

En bref, un programme résoudrait un puzzle en plaçant le chiffre “1” dans une cellule et vérifie s'il est autorisé à s'y trouver.

- S'il n'y a aucun problème , l'algorithme passe à la cellule suivante et y place un “1”.



```
paul at kolowy in ~/Documents/S3/ocr/Sigma-Vision/src/solve
$ cat grid_00
... .4 58.
... 721 ..3
4.. ... ...
...
21. .67 ..4
.7. ... 2..
63. .49 ..1
...
3..6 ... ...
... 158 ..6
... ..6 95.

paul at kolowy in ~/Documents/S3/ocr/Sigma-Vision/src/solve
$ ls
grid_00  solver

paul at kolowy in ~/Documents/S3/ocr/Sigma-Vision/src/solve
$ ./solver grid_00

paul at kolowy in ~/Documents/S3/ocr/Sigma-Vision/src/solve
$ ls
grid_00  grid_00.result  solver

paul at kolowy in ~/Documents/S3/ocr/Sigma-Vision/src/solve
$ 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
```

FIGURE 18 – Résultat final !

- Si l'on découvre que le “1” n'est pas autorisé, la valeur est avancée à “2”.
- Si l'on découvre une cellule où aucun des 9 chiffres n'est autorisé, l'algorithme laisse cette cellule vide et revient à la cellule précédente. La valeur de cette cellule est alors incrémentée de un. Cette opération est répétée jusqu'à ce que la valeur autorisée soit découverte dans la dernière (81e) cellule.

Un **problème** peut être lié à :

- contraintes de ligne
- contraintes de colonne
- contraintes de case



## 8 Reconstruction de l'image

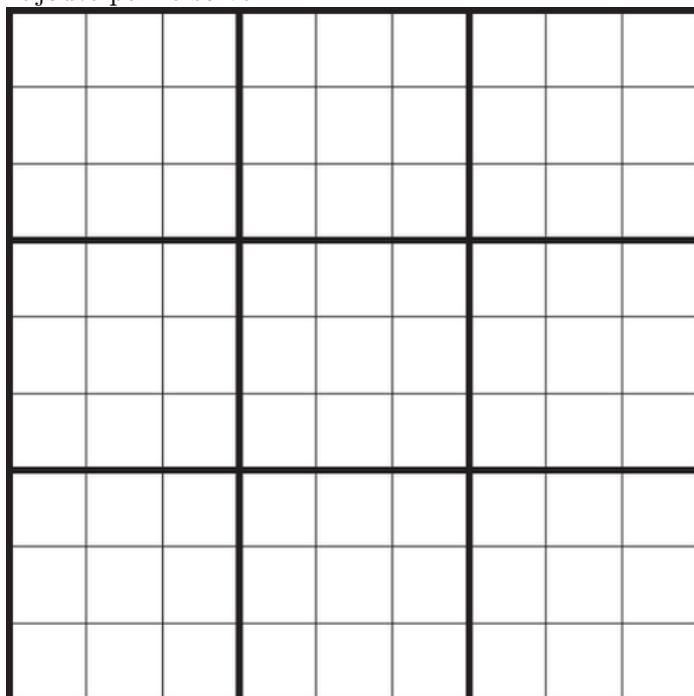
Quand le solver a fini son boulot, vient le moment de la reconstruction de la grille. Pour cela, nous avons opté pour la librairie SDL. Cette dernière sera utilisée dans une fonction qui prend en paramètre 2 fois la même grille, l'une non résolue et l'autre résolue.

### 8.1 Lecture des fichiers grid

Pour comprendre la grille qui doit être reconstruite, la fonction viendra ouvrir les fichiers grid00 et grid00.result (deux fichiers contenant la même grille mais dans le .result, cette dernière est résolue), lire caractère par caractère en éliminant les espaces et retours à la ligne inutile. Ces grilles seront sauvegardées dans 2 arrays différents.

### 8.2 Association des Surfaces

Pour commencer, la fonction va créer une SdlSurface à partir d'un png de grille complètement vide contenu dans le dossier /resources. Ensuite, elle parcourra les arrays des grilles en parallèle, et pour chaque nombre, elle créera soit une SdlSurface du nombre en noir ou en rouge à partir de png eux aussi contenu dans le dossier /resources (en noir si le nombre est contenu dans les 2 arrays et déjà présent sur la grille et en rouge si le nombre est seulement contenu dans l'array associé à la grid00 résolue et un nombre rajouté par le solve).



La grille complètement vide



Exemple de png associé à chaque nombre

### 8.3 Construction de la surface finale

Pour finir, on va appeler la fonction Sdl\_BlitSurface qui viendra coller les surfaces qu'on avait créées avec les arrays sur la surface contenant la grille vide en lui donnant les bonnes coordonnées à chaque fois. Une fois tout cela fini, nous libérons les espaces mémoire associés aux arrays et aux surfaces pour

les nombres et l'on renvoie la surface contenant du coup la grille remplis avec tous les nombre. Elle devrait ressembler a cela :

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

Grille résultante de la reconstruction de la grille



## 9 Regrouper le projet

Il fallait ensuite s'occuper de regrouper tout le projet.

Pour cela, il a fallu modifier chaque fichiers principal afin qu'il prenne une surface SDL en entré et une surface SDL en sortie. Une fois cela fait, un grand makefile permet de tout compiler. En effet, tout les fichiers .c sont compilés ensemble. Il a aussi fallu faire en sorte que tous les dossiers ne compilent pas afin d'éviter les fonctions main.

La commande pour prendre tout les fichiers est donc :

```
““ SRCS := $(shell find(SRC_DIRS) -name ‘*.c’ -not -path “*/no-make/*”) ““
```

De plus, il a fallu faire en sorte que le dossier */ressources/* soit à la racine du projet et contienne tout les fichiers pour l'UI.

Le Makefile va générer un dossier */build/* qui contiendra tous les fichiers '.d' et '.o' des fichiers compilés. Il fait aussi un exécutable à la racine du dossier *build*

## 10 Management

Nous avons utilisé github dashboard pour le management du projet. Cela nous a permis d'avoir un suivi en temps réel sur les problèmes que les différents membres du groupe rencontraient, et donc de pouvoir s'entraider si le besoin en venait. Voici un screenshot que nous avons eu l'occasion de prendre plus tard dans le déroulement du projet.<sup>20</sup>

## 11 Mirroring

Nous avons essayé de faire du mirroring du projet sur gitlab, mais nous n'avons pas réussi.

```
paul at kolowy in ~/Documents/S3/ocr/S  
$ tree build  
build  
└── final_program  
    ├── src  
    │   ├── main.c.d  
    │   ├── main.c.o  
    │   └── neural  
    │       ├── config_manager.c.d  
    │       ├── config_manager.c.o  
    │       ├── digit_guess.c.d  
    │       ├── digit_guess.c.o  
    │       ├── digit_learn.c.d  
    │       ├── digit_learn.c.o  
    │       ├── nn_tools.c.d  
    │       ├── nn_tools.c.o  
    │       ├── test.c.d  
    │       └── test.c.o  
    └── xor_entrain...  
        ├── config_manager.c.d  
        ├── config_manager.c.o  
        ├── nn_tools.c.d  
        ├── nn_tools.c.o  
        ├── xor.c.d  
        ├── xor.c.o  
        ├── xor_learn.c.d  
        ├── xor_learn.c.o  
        ├── xor_use.c.d  
        └── xor_use.c.o  
    └── preproc  
        ├── neutralize.c.d  
        ├── neutralize.c.o  
        ├── preproc.c.d  
        ├── preproc.c.o  
        ├── rotate.c.d  
        └── Structedrotate.c.o  
    └── rotate  
        ├── scale.c.d  
        ├── scale.c.o  
        ├── tools.c.d  
        ├── tools.c.o  
        ├── transform.c.d  
        └── transform.c.o  
    └── rebuild  
        ├── rebuild.c.d  
        └── rebuild.c.o  
    └── solve  
        ├── solver.c.d  
        └── solver.c.o  
    └── UI  
        ├── ui.c.d  
        └── ui.c.o
```

FIGURE 19 – Dossier build généré par le makefile



The screenshot displays the Sigma Vision project management interface with three main sections: Suggests, Todo, and Doing.

**Suggests (5 items):**

- project-management #9 ...  
Determine better activation function  
Very low enhancement neural network  
off code question
- project-management #24  
find a way for 12 cells sudoku  
bug enhancement help wanted
- project-management #25  
Gérer les erreurs et Exception proprement  
Low enhancement
- project-management #27  
Gestion de la convexité  
High preprocess  
Rendu Soutenance Finale
- project-management #31  
sobel probleme

**Todo (4 items):**

- project-management #19  
Write XOR part documentation (README + doc-hugo)  
Low documentation neural network  
Point hebdomadaire #4
- project-management #20  
Improve binarization  
Low preprocess
- project-management #22  
Use xor network  
High code block neural network  
Point hebdomadaire #3
- project-management #13  
Hough transform  
Medium preprocess

**Doing (3 items):**

- project-management #28  
Rotation by area mapping  
Low preprocess
- project-management #29  
Grid Cropping  
High preprocess  
Rendu Soutenance intermédiaire
- project-management #30  
find grid

FIGURE 20 – Project Management



## 12 Conclusion

Nous estimons avoir rempli les minima attendus pour cette dernière soutenance et sommes tout de même satisfait par le projet dans son ensemble, malgré que certains objectifs personnels n'aient pas été atteints.

Chaque partie est totalement fonctionnelle, même si certaines améliorations mineures pourraient être apportées.

Le prétraitement de l'image est performant et est largement assez satisfaisant pour la grande majorité des images.

Le réseau neuronal est totalement fonctionnel, il était déjà très bien avancé pour la première soutenance donc cela a été relativement aisément de le terminer.

L'interface utilisateur, qui était déjà fonctionnelle à la première soutenance, s'est développé avec plusieurs fonctionnalités intéressantes, telles que le dark mode et la reconstruction d'image.

Le programme résolvant le sudoku est toujours fonctionnelle et n'a pas changé depuis la dernière soutenance.

Nous avons pu, depuis la dernière soutenance, nous pencher sur chaque détail de notre projet, et avons pu rejoindre nos parties ensemble. Malgré certains petits conflits entre les entrées et les sorties, la jonction s'est faite sans problèmes, et nous avons pu ainsi profiter du fait de pouvoir travailler indépendamment les uns des autres, et cela nous a permis de nous éloigner totalement du domaine des autres membres du groupe, et ainsi approfondir le notre.

Nous sommes donc somme toute assez satisfait de ce projet et avons atteint notre objectif principal qui était d'avoir un OCR fonctionnel.