SIGMA VISION

Eloi COUDERC Paul DUFOUR Malo GARNIER Josef TOURRAND

1 Introduction

1.1 Présentation du projet

Ceci est le rapport de la première soutenance de notre projet de troisième semestre. Pour présenter vite fait le projet, il faut réaliser un OCR (*Optical Character Recognition*) dont le but est de reconnaître les chiffres que composent un Sudoku de 3 par 3 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 faire un point écrit sur notre avancement, notre répartition du travail jusqu'ici, notre organisation, etc... Il nous permet aussi d'exprimer les objectifs que nous nous sommes fixé pour ce projet.
Nous finissons cette introduction en vous souhaitant une excellente et heureuse lecture !

Table des matières

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 neuronal (création, apprentissage et sauvegarde des poids).
- **Paul Dufour**, qui s'est chargé de la détection de la grille et du solver.
- **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 de génération d'image d'entraînement pour le réseau neuronal.

2.2 L'organisation

Dès le début du projet, un serveur Discord et une page GitHub ont été crée, 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'a 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 facile, mais toujours utile.

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 Attendus, objectifs et réalisations

Il est requis pour cette première soutenance de disposer d'un réseau réglé pour résoudre la fonction logique XOR. Notre volonté va plus loin car nous souhaitons disposer d'un système facilement adaptable afin de pouvoir le convertir rapidement pour la reconnaissance de chiffres.
Nous avons ainsi développé un système entièrement configurable et universel fonctionnant avec n'importe quel configuration de réseau de neurones. Pour cette soutenance, le réseau a été adapté pour la fonction XOR.

3.2 Fonctionnement

Le programme est divisé en deux parties :
- l'apprentissage
- l'utilisation

Un seul exécutable est généré et l'utilisateur choisit le mode en passant le paramètre `-l` (learn) suivi du nombre d'itérations ou `-u` (use) suivi des deux entrées (0 ou 1). Si aucun fichier de configuration (cf *Sauvegarde des poids*) n'est présent, le réseau doit être entrainé pour en générer un. Le mode utilisation affiche le résultat sur la sortie standard.

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

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 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 réentrainer 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 :

3.4.1 SUMMARY

Le nombre de couches est indiqué sur la première ligne.

```
l <nb>
```

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

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

3.4.2 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> [...]
```

3.4.3 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

3.5 Perspectives futures

Nous avons développer une structure de réseau de neurones universelle donc l'adaptation à la reconnaissance de chiffres sera rapide. Pour cela, nous devrons juste modifier le nombre de couches et de nœuds ainsi que le dataset d'entraînement.
Afin d'améliorer la précision du réseau, nous comptons utiliser une meilleure fonction d'activation. L'actuelle est une sigmoïde mais nous pensons la remplacer par une Unité exponentielle linéaire par exemple.

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 a la conclusion qu'il serait préférable de coder nous même une générateur de .png. Pour cela, nous avons écris un petit code sur python qui génère un nombre prédéfini dans le code d'image dans un dossier puis ajoute chacune de ses dernière 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).

```python
from PIL import Image, ImageDraw, ImageFont
from zipfile import ZipFile
import random, os, shutil

NUMBER_OF_IMAGE = 150

#Check if the folder results exist
if os.path.exists("results"):
    shutil.rmtree("results", ignore_errors=False, onerror=None)

os.mkdir("results")

#Generate image with random numbers and font. Save them into the result folder
for i in range(NUMBER_OF_IMAGE):
    Fpath = "font/" + random.choice(os.listdir("font\\"))

    img = Image.new('RGB', (15,23), color = (255, 255, 255))
    fnt = ImageFont.truetype(Fpath, 20)

    d = ImageDraw.Draw(img)
    d.text((random.randint(-2,6),random.randint(-4, -2)), str(random.randint(0, 9)), font=fnt, fill=(0, 0, 0))

    a = random.randint(0,2)
    if (a == 0):
        img = img.resize((8, 8))
    elif (a == 1):
        img = img.resize((16, 16))
    else:
        img = img.resize((32, 32))
    img.save("results/" + str(i) + ".png")

#Create a zipfile archive and put all the result in it
with ZipFile('numberTest.zip', 'w') as zipObj:
    arr = os.listdir("results\\")
    for file in arr:
        zipObj.write("results/" + file)

#Remove the results folder afterwards
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* ou 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 a un emplacement aléatoire sur un font 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 répété n fois dépendamment de la variable NUMBER_OF_IMAGE.
Une fois toute les images crée, 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 généré par le code

5 Traitement d'image

Le traitement d'image a pour but de regler tous les problemes liés a l'image. Les problèmes imaginable 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 etre très lisible, ou flous et donc complexifier la tache de la reconnaissance de chiffres.
- **Les déformations** les surfaces ne sont pas forcement droites, et donc peuvent poser problème. Les grilles peuvent aussi être pliés.

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

- **Grayscale**
- **Flou Gaussien**
- **Binarisation**
- **Détection de bords**
- **Détection de la grille**
- **Correction de la rotation**
- **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.

5.2 Flou Gaussien

Le **Flou Gaussien** permet d'éliminer les petites imperfections de la grille 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 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.

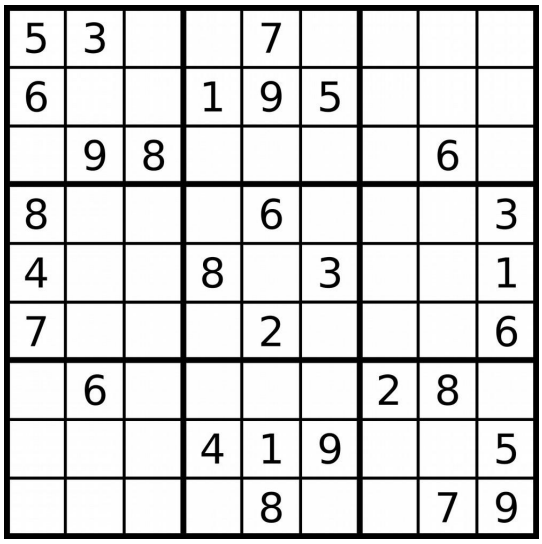


FIGURE 2 – Image avant l'application du Flou Gaussien

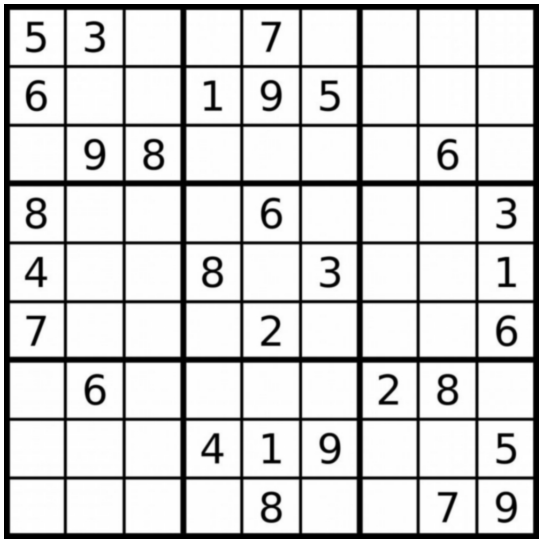


FIGURE 3 – Image après l'application du Flou Gaussien

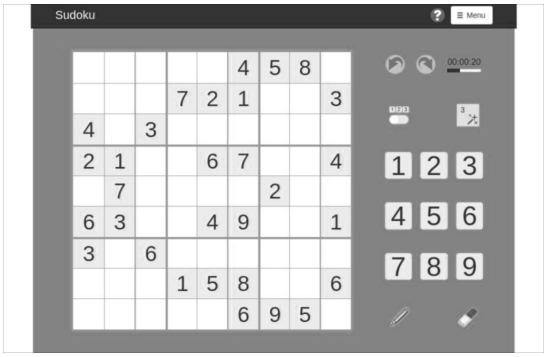


FIGURE 4 – Image avant l'application de la binarisation d'Otsu

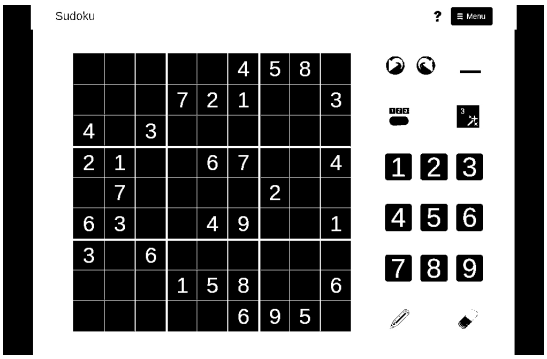


FIGURE 5 – Image après l'application de la binarisation d'Otsu

5.3 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. C'est pourquoi nous avons décidé d'implémenter la **binarisation d'Otsu**, avec une normalisation de l'histogramme.
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 etre considérés comme des pixels appartenant au premier plan. L'algorithme de la binarisation d'Otsu va chercher a maximiser une variance, et la valeur ayant la plus grande variance est celle qui sera choisie en tant que valeur seuil.
Comme l'on peut le voir dans les figures 4 et 5, la binarisation d'Otsu permet de neutraliser grandement l'image, et rend donc tout le reste du pretraitement de l'image bien plus simple.

5.4 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

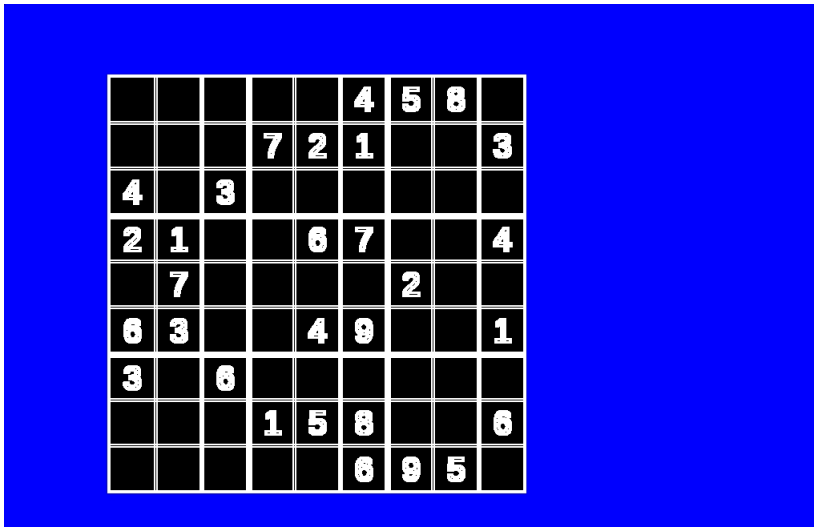


FIGURE 6 – Image apres l'application du processus pour trouver la grille

structure pour retenir les différents points.

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

typedef struct Square
{
    Dot topLeft;
    Dot topRight;
    Dot bottomLeft;
} 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.

5.5 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. On applique ensuite la formule suivante sur les coordonnées de ces 2 points, afin de déterminer l'angle de rotations en radians. Soient (x1,y1) les coordonnées du point en haut à gauche et (x2,y2) les coordonnées du point en haut à droite :

$$-arctan(\frac{(y2-y1)}{(x2-x1)})$$

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

$$xp = (x - cx) * cos(a) + (y - cy) * sin(a) + cxyp = -(x - cx) * sin(a) + (y - cy) * cos(a) + cy$$

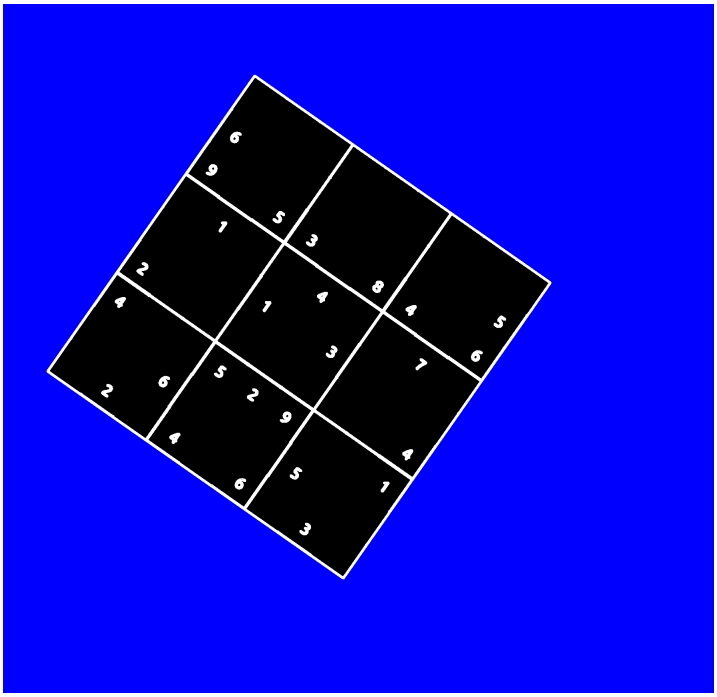


FIGURE 7 – Autre image apres l'application du processus pour trouver la grille

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 peuvene

5.6 Découpage de l'image

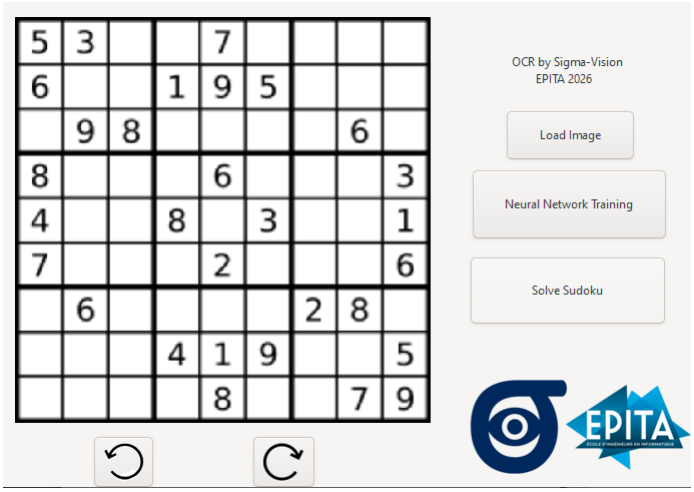
L'image doit enfin être découpé en une image par cases, afin de l'envoyer au réseau de neurone. Pour cela nous réutilisons les coordonnées du square précèdent.

6 Interface Utilisateur

Pour la première soutenance, nous nous sommes 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'a coder tout le back avec GTK.

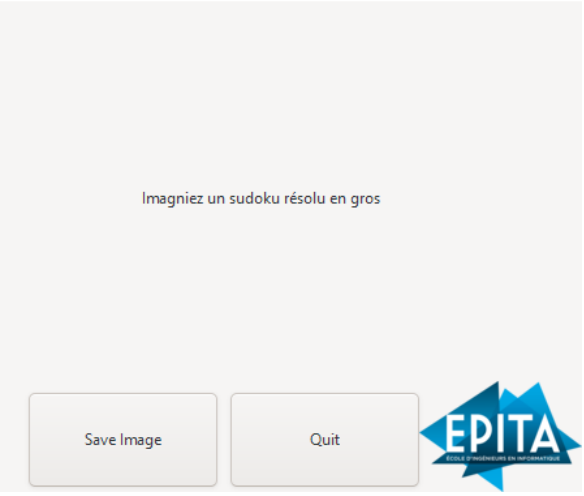
Actuellement, l'interface est divisé en 3 menu :

— Le menu principal, comportant plusieurs boutons, notamment celui du chargement de l'image, celui d'accès aux autres menu et ceux qui serviront de correction manuelle pour la rotation au cas ou la rotation automatique ne serait pas parfaite. De plus, lorsqu'une image est chargé grâce au bouton, elle sera resize pour l'affichage de façon a ce qu'elle rentre dans le carré qui lui est dédié et s'affichera a gauche de l'interface. Actuellement, les boutons de chargement d'image et d'accès aux autres menu sont fonctionnels. Ceux de la rotation ne sont pas encore implémenté.



— Le menu d'entraînement du réseau neuronal. Celui-ci servira d'interface oui l'on pourra déposer les images que l'on souhaitera utiliser lors de l'entraînement du réseau de neurone. Il sera composé d'une liste qui se remplira avec les images que l'ont ajoutera a l'entraînement et de plusieurs boutons en bas, nottament celui du chargement des images, celui qui sauvegarda le choix des images et celui qui permet de quitter le menu. Ce menu n'a actuellement aucune fonctionnalité sauf le bouton QUIT.

— Le menu de la grille résolue. Cette dernière affichera la grille de sudoku en gros et permettera la sauvegarde de cette dernière en image sous le format .png. Les deux boutons présent sur ce menu sont tous deux fonctionnels (celui de sauvegarde de l'image ouvre une fenêtre ou l'on peut choisir un fichier de son ordinateur ou l'on veut que l'image soit sauvegardée).



Actuellement, l'interface n'est pas relié au reste du projet. Elle est juste une esquisse de l'UI qui sera présenté lors de la dernière soutenance. C'est pour cela que la mise en page des différents menu risque d'être modifié, et il est probable que certains menu soit fusionné entre eux. Nous comptons aussi ajouter des éléments graphique qui offrira une identité a l'application.

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 ficher **$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 ficher sortira un ficher de nom **$(name).result**.
Le fichier respectera les mêmes spécifications. Le résultat du fichier précèdent 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”.

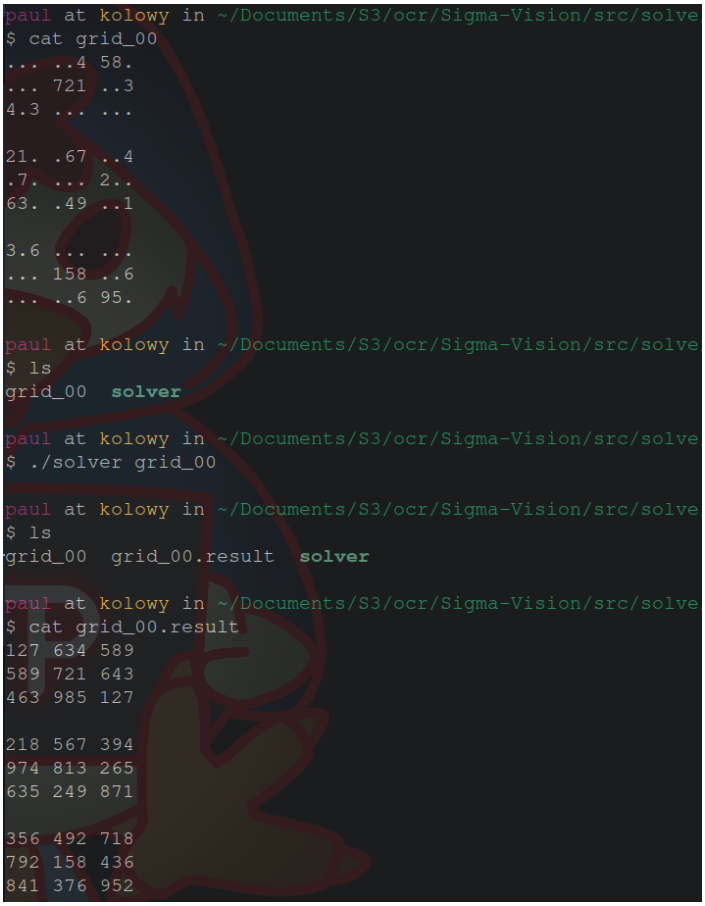


FIGURE 8 – Résultat final !

— 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 Conclusion

Nous estimons avoir rempli les minima attendus pour cette première soutenance et nous sommes même bien avancés sur plusieurs points.
La partie liée au réseau de neurone est à son état quasiment final, il ne reste plus qu'à modifier les paramètres pour déterminer efficacement les chiffres mais la structure générale est aboutie.
Pour ce qui est du prétraitement de l'image, nous avons des résultats déjà satisfaisants et nous sommes en train de développer les derniers éléments.
L'interface utilisateur est prête et complète. Elle permet d'effectuer l'ensemble des actions nécessaires de manière intuitive et efficace.
Le programme résolvant le sudoku est fonctionnel.
Le découpage de la grille est en cours, il reste quelques bugs à résoudre.

Nous avons rédigé la structure globale de notre projet et son architecture, il nous faut désormais développer les derniers éléments restants. Nous avons dès le début axé notre façon de travailler sur l'atomisation des tâches de sorte à ce qu'elles puissent être développées indépendamment tout en s'assemblant sans problème. Chaque personne travaillant sur un point connaît ce qu'elle prend en entrée et ce qu'elle doit renvoyer. Ainsi, lors de la mise en commun nous n'avons plus que des blocs à assembler entre eux.
Nous comptons avoir un OCR fonctionnel dans les prochaines semaines afin de pouvoir l'améliorer par la suite et rajouter des fonctionnalités satellites.