Sodoko

GOPY

Rapport de soutenance n°2

Grégoire Lefaure Oscar Le Dauphin Yness Khouader Pierre Wagniart

11 décembre 2022

Introduction

Bonjour et bienvenue dans le deuxième et dernier épisode de « Sodoko, l'OCR à tous prix ».

Dans cette épisode, nous allons vous représenter rapidement ce qui était prêt et fonctionnel lors de la première soutenance, mais nous allons surtout vous expliquer ce qui a changé.

Comme les changements sont nombreux et généraux, nous allons suivre le même type de plan que lors de notre premier rapport de soutenance car il suit la structure de notre code, et surtout les différentes étapes de la totalité du traitement effectué par notre programme.

Vous pouvez voir le plan via la table des matières sur la page suivante.

Nous n'allons pas représenter une seconde fois ce qu'est un OCR, et nous allons donc vous laisser continuer avec nous en vous souhaitant une bonne lecture.

Table des matières

1 Qui sommes nous ? 6

1.1 Grégoire Lefaure . . . 6

1.2 Oscar Le Dauphin . . . 6

1.3 Yness Khouader . . . 6

1.4 Pierre Wagniart . . . 6

2 Développement du programme 7

2.1 Nos outils pour développer en équipe et en toute sérénité . . . 7

2.1.1 Discord . . . 7

2.1.2 Github . . . 7

2.1.3 Criterion . . . 8

2.1.4 Notion . . . 8

2.2 Glade . . . 9

2.3 Le développement commun . . . 10

2.3.1 L'architecture du projet . . . 10

2.3.2 Vive les `Makefile` . . . 10

2.3.3 Notre boite à outils . . . 11

2.4 Le pré-traitement . . . 12

2.4.1 Elles sont gigantesques ! . . . 12

2.4.2 255 nuances de gris . . . 12

2.4.3 Aie, j'ai mal aux yeux ! . . . 12

2.4.4 Il fait tout noir ! . . . 15

2.4.5 Mais où est la grille ? . . . 16

2.4.6 Tournez les images ! . . . 16

2.4.7 Mais c'est pas droit cette histoire ! . . . 17

2.4.8 Découpons-les... . . . 17

2.4.9 Sont-elles de la bonne taille ? . . . 17

2.4.10 De la couleur à la reconnaissance... . . . 18

2.5 Le réseau de neurones . . . 19

2.5.1 Optimisations . . . 19

2.5.2 Résultats . . . 20

2.5.3 Jeu de données . . . 21

2.6 La résolution . . . 22

2.6.1 Et si nous chargions la grille ? . . . 22

2.6.2 Résolvons-la ! . . . 22

2.6.3 Enregistrement en cours... . . . 23

2.6.4 Ça y est, on a trouvé la solution ! 24
2.7 Le post-traitement 25
2.7.1 Récupération du résultat 25
2.7.2 Création de la grille 25
2.7.3 Affichons la résolution 26
2.8 La GUI pour les petits 28
2.8.1 Fonctionnalités 28
2.8.2 Une animation 29
2.8.3 Des technalités... 32
2.9 C'est déjà la fin... 32

3 Des petits plus **33**
3.1 Un programme en ligne de commande 33
3.2 Un site internet 34

4 Notre ressentis sur le projet **36**
4.1 Grégoire Lefaure 36
4.2 Oscar Le Dauphin 38
4.3 Yness Khouader 38
4.4 Pierre Wagniart 39

5 Conclusion **40**

1 Qui sommes nous ?

Nous allons passer rapidement sur cette sections, car nous nous sommes déjà présentés en détails lors du premier rapport de soutenance. Nous allons donc principalement rappeler les parties sur lesquelles nous avons travaillé pour que vous puissiez plus facilement vous situer dans la suite de ce rapport.

1.1 Grégoire Lefaure

Hello hello ! Moi c'est Grégoire, et je suis le chef de projet de notre groupe. Je me suis principalement occupé du pré-traitement, un tout petit peu du solver pour le jeu de tests, ainsi que de l'organisation générale de l'architecture (Makefile, architecture du projet, etc.). Je n'ai bien entendu pas fait tout ça tout seul, j'ai été aidé des autres membres de mon groupe pour mener ces tâches à bien.

1.2 Oscar Le Dauphin

Bonjour ! Moi c'est Oscar, et je me suis occupé du réseau de neronnes pour la reconnaissances des chiffres, et de la detection de la grille. Une grande partie de mon temps a été consacrée à la création du jeu de données pour entrainer le réseau de neronnes. Je dois avouer que trier des chiffres pendant des heures n'était pas très amusant... Mais le résultat en a value le coût.

1.3 Yness Khouader

Hey ! Moi c'est Yness, le Y de GOPY. Au cours de ce projet j'ai pu m'occuper dans un premier temps du programme de résolution de sudoku. Par la suite, après l'élaboration des croquis j'ai tenté de contribuer à la construction de l'interface graphique et du code en gtk permettant de rendre le tout interactif. Néanmoins j'ai eu beaucoup de mal avec cet outil j'ai donc très peu assisté mes camarades sur cette tache. Enfin, je me suis chargée de la conception du site internet.

1.4 Pierre Wagniart

Bien le bonjour, je suis le seul, l'unique et le petit Pierre Wagniart. Je me suis chargé de quelques fonctions du preprocess ainsi que d'une grande partie du GUI. Je me permet d'utiliser cette section pour remercier le chef de projet sans qui l'organisation aurait été bien plus fastidieuse.

2 Développement du programme

2.1 Nos outils pour développer en équipe et en toute sérénité

2.1.1 Discord

Commençons par l'outil qui nous est le plus indispensable : Discord. C'est en effet ce moyen de communication que nous avons choisi pour nous permettre de communiquer et surtout d'organiser notre communication en différents *channels*. Notre utilisation de cet outil n'a pas beaucoup évolué depuis la première soutenance, c'est pourquoi nous ne détaillerons pas plus son utilité dans notre projet ici.

2.1.2 Github

Le second outil qui est lui aussi indispensable à la construction de ce projet est Github. En effet, c'est cette plateforme que nous utilisons pour héberger notre code, faire des *Pull Request*, des commentaires, etc... Nous n'utilisons en effet pas le répertoire `git` fourni par l'école, car nous le trouvons assez limité par rapport aux fonctionnalités de Github (Pull Request, Continious Integrations, etc...).

Sur le même principe que pour la première soutenance, un système de texttt« mirroring » *fait maison* est toujours en place. Nous l'avons un peu amélioré puisqu'au lieu de devoir le faire manuellement une fois de temps en temps, nous avons automatisé cette tâche grâce à Linux, et le répertoire distant du cri est maintenant mis à jour tous les jours à 4h du matin. Nous avons même prévu une activité plus intense à l'approche de la deadline puisque sur la dernière semaine, le répertoire distant du cri est mis à jour toutes les heures, et même tous les 5 minutes pour le dimanche soir entre 19h et 20h, pour être certains que nos changement de dernière minute soient pris en compte dans le rendu de notre projet.

Comme dit précédemment, nous utilisons donc beaucoup les `Pull Request`[1]. Nous avons continué d'utiliser ce système de fonctionnement. Il nous permet en effet de nous tenir au courant plus facilement du code produit par les autres et que nous allons probablement devoir utiliser un jour ou l'autre.

Concernant les `Continuous Integrations`, elles n'ont pas beaucoup changées puisqu'elles se basent toujours sur le même principe, et nous n'en n'avons pas rajoutés de nouvelle. Nous nous sommes en effet contenté de les mettre à jour en fonctions des nouveaux paquets dont nous avions besoin.

Héberger notre code sur Github nous permet aussi de rendre public notre projet, non pas pour que d'autres l'utilisent (vouloir résoudre un sudoku via OCR n'est pas forcément un problème quotidien), même si ce n'est pas impossible pour autant. Non, la portée publique de

1. Que nous abrégerons en « PR »dans la suite de ce rapport

notre projet s'inscrit surtout dans volonté de maintenir un code le plus « open source »possible.

2.1.3 Criterion

Pour tester notre code, nous avons choisi l'infrastructure logiciel (framework) de tests Criterion. Ces tests sont executés sur un serveur nous appartenant à chaque fois que l'un d'entre nous push son code. Cela nous permet de rapidement repérer quel est le commit dans une branche qui pose problème, puisqu'il y aura une croix à côté indiquant que nos tests de sont pas passés.

En plus de très bien s'intégrer avec Github, écrire des tests permet une expérience de développement bien plus rapide et agréable. En effet, la technique du `développement piloté par les tests`[2] est très répandue pour les projets de grande ampleur. Elle consiste à écrire les tests unitaires avant les fonctions en elle-mêmes. L'interêt de travailler de cette manière est que nous pouvons avancer plus vite, quand une fonction passe ses tests, on passe à la prochaine, quand on modifie une ancienne fonction, on vérifie directement que les tests passent toujours. Ainsi, en plus d'un développement plus agréable, on ne risque pas de régresser dans l'avancement du projet en cassant des fonctions préalablement écrite sans s'en rendre compte.

Nous avions beaucoup utilisé cette technique de tests pour le solver, comme vous allez le voir par la suite, ainsi que pour le réseau de neurones, mais malheureusement, cette technique de tests atteint rapidement ses limites dans le cas d'un OCR. En effet, ces tests sont quasiment totalement inutiles pour le pré-traitement, étant donné que nous voulons un résultat utilisable, mais qu'il n'est pas forcément le même. De plus, ce n'est pas vraiment soit tout bon, soit tout faux. Lorsque nous sommes sur la bonne piste, le résultat n'est pas parfait pour autant et nous devons encore modifier quelques paramètres pour que cela fonctionne vraiment bien. Des tests unitaires comme ceux proposés par Criterion ne nous sont donc pas d'une grande utilité dans le cadre des fonctions de pré-traitement.

2.1.4 Notion

Notion est le dernier outil que nous utilisons souvent pour mener à bien notre projet. C'est un outil collaboratif permettant d'écrire des pages (un peu comme un site internet) de manière collaboratives, avec quelques fonctions intéractives très pratiques comme des rappels, une organisation claire des tâches, etc... La partie publique de ce Notion est d'ailleurs disponible ici : `https://gopy.notion.site`, pendant la durée du projet.

Dans la partie privée du Notion, nous pouvons retrouver quelques ressources qui nous sont utiles pour notre projet (le cahier des charges, pour ne pas avoir besoin de le rechercher dans

2. (ou `test driven development` en anglais)

moodle à chaque fois, etc...). Nous pouvons aussi retrouver une page spéciale dédiée à l'avancement du projet avec une todo list avec la ou les personnes s'occupant de telle ou telle tâche, des notes ou remarques sur la tâche en question, ainsi qu'une deadline pour une tâche donnée.

En ce qui concerne la partie publique de ce site web, elle concerne surtout la documentation de notre projet. Il se divise en trois parties disctinctes.
La première partie contient les pages classiques d'un projet : présentation de l'équipe, comment installer notre programme, un lien vers notre Github, ainsi que des pages donnant quelques tips que nous avons utilisé pour développer notre OCR (des alias git, des configurations vim, etc...).

La seconde partie est composée de *How to*, c'est-à-dire que ce sont des pages qui nous guident dans les différentes étapes de notre projet.
Enfin, la dernière partie, et la plus importante, est la documentation de notre code. En effet, nous souhaitons documenter notre code un maximum pour nos co-équipier, afin qu'ils puissent comprendre comment telle ou telle partie du programme fonctionne, ou même comment utiliser nos fonction dans une autre partie du programme.

2.2 Glade

L'idée de placer chaque bouton manuellement dans le code, coordonnées par coordonnées, ne nous enchantait guère. Heureusement nous avons découvert Glade, mais qu'est-ce ? Glade est un outil interactif de conception d'interface graphique GTK. Il prend en charge toute la partie de gestion/génération de l'interface pour permettre au développeur de se concentrer sur le code « utile ». Glade enregistre les interfaces graphiques en générant des fichiers XML. Initialement la bibliothèque libglade permettait de lire ces fichiers dynamiquement (c'est-à-dire à l'exécution de l'application). GtkBuilder (inclus dans GTK) remplace dorénavant libglade dans l'environnement de bureau GNOME. Ces fichiers XML peuvent être utilisés par de nombreux langages de programmation tels que C++, Python, JavaScript, Vala, Haskell, Rust et FreeBasic et celui qui nous intéresse ici : C.

2.3 Le développement commun

2.3.1 L'architecture du projet

Nous avons mis en place une architecture claire permettant une meilleure organisation de notre projet. Notre principal ajout depuis la première soutenance est l'ajout d'un dossier `assets`. Il nous permet de stocker toutes les resources nécessaires au bon fonctionnement de notre programme. Voici un screenshot de l'architecture finale de notre projet :

```
.
|-- assets
|   |-- GUI
|   |-- NeuralNetwork
|   |-- font
|   `-- template
|-- src
|   |-- CLI
|   |-- GUI
|   |-- NeuralNetwork
|   |-- Postprocess
|   |-- Preprocess
|   |-- Solver
|   `-- Utils
|-- tests
|   |-- NeuralNetworkTests
|   |-- PreprocessTests
|   |-- SolverTests
|   |-- UtilsTests
|   `-- images
`-- website
```

2.3.2 Vive les `Makefile`

Cette étape est une partie importante de notre projet. Nous avons d'abord fait un Makefile très basique en se basant sur un template trouvé sur internet, mais plus nous avancions dans le projet, moins ces Makefile étaient adaptés à notre usage. C'est pourquoi nous avons restructuré nos Makefile presque entièrement afin de correspondre pleinement à nos besoins. Avant de commencer à expliquer comment nos Makefile fonctionnent, nous tenions à remercier nos ASM pour leur conférence sur le fonctionnement des Makefile qui nous a été extrêmement utile.

Nous avions déjà expliqué la structure de nos Makefile dans le précédent rapport de soutenance, et elle est restée quasiment la même depuis la première soutenance. Faisons malgré tout un petit rappel pour mieux comprendre leur utilisation depuis la première soutenance.

Nous avons un Makefile principal à la racine de notre projet qui nous permet de compiler entièrement notre projet. Les fichiers de compilations sont placé dans un dossier spécial nommé `_build/`, dont l'architecture interne suit celle de notre projet avec tous les fichiers `.o` et `.d`.

Nous avons ensuite un autre Makefile qui nous permet de compiler seulement certaines parties de notre projet (uniquement le `Preprocess`, le `Solver` ou autre). Ces Makefile nous permettent donc de tester manuellement notre code sans pour autant compiler tout le projet.

Enfin, notre dernier Makefile est notre Makefile situé dans `tests/`, et, comme son nom l'indique, nous sert à exécuter des tests automatiques sur notre projet. Si ce n'est pas déjà fait, il appelle le Makefile principal pour compiler tout le projet, puis compile ensuite les tests avant de les exécuter.

Ce sont le Makefile principal et le Makefile de tests qui sont utilisés dans les CI pour tester notre code avant de le merge sur la branche principale de notre répertoire distant.

Les modifications principales depuis la première soutenance résident surtout en l'ajout des librairies dont nous avions besoin pour continuer notre projet.

De plus, dans le but de simplifier et centraliser les paramètres de compilation que nous utilisons, nous avons créé un fichier `compile_settings.mk`. Ainsi, nous n'avons plus besoin de modifier les paramètres de compilation dans chacun des Makefile, mais seulement dans ce fichier qui est ensuite inclus dans chacun des Makefile.

Nous utilisons aussi un fichier `pretty_print.mk` qui nous permet d'afficher un message coloré en fonction du résultat de la compilation. Ainsi, nous avons une jolie suite d'instructions vertes au lieu d'une suite monotone de lignes de commande blanches et ternes. Nous voyons ainsi directement si notre code a un problème !

2.3.3 Notre boite à outils

Depuis la première soutenance, nous avons ajouté énormément de fonctions dans le dossier `Utils` de notre architecture. Nous n'allons pas toutes les détailler ici puisque ça ne serait pas forcément très utile, mais nous allons tout de même mettre un screenshot des différents fichiers, qui, nous l'estimons, possèdent un nom assez explicite pour ne pas avoir besoin de détailler le rôle de chacun d'entre eux :

```
→  Utils git:(main) ls
Makefile      array_sort.h  maths.h                matrice_multithread.h  point.c         queue.c      str_utils.c
array.c       file_utils.c  matrice.c              matrice_type.h         point.h         queue.h      str_utils.h
array.h       file_utils.h  matrice.h              pixel_utils.c          progress_bar.c  sdl_utils.c
array_sort.c  maths.c       matrice_multithread.c  pixel_utils.h          progress_bar.h  sdl_utils.h
```

2.4 Le pré-traitement

2.4.1 Elles sont gigantesques !

Cette étape n'était pas présente lors de notre première soutenance, et pourtant elle nous a sauvé d'une grosse panade. En effet, dans le but d'harmoniser et d'homogénéiser les images, nous avons décidé de redimensionner toutes les images passées en entrée du programme pour qu'elles fassent toutes la même taille : 1000 pixels par 1000 pixels. L'intérêt de cette étape est qu'elle nous permet, par exemple lors de l'utilisation de filtres, d'utiliser une taille fixe et connues lors de la compilation afin de simplifier notre code. Cela permet aussi de ne pas avoir besoin de calculer à chaque fois la taille des filtres que nous utilisons. En effet, la taille de ces filtres par rapport à la taille de l'image influe énormément sur la qualité des résultats des différentes étapes de l'entièrete de notre pré-traitement.

2.4.2 255 nuances de gris

Seconde étape du prétraitement, elle est donc très importante. Ainsi, elle permet de simplifier tous les algorithmes suivants, non seulement dans le pré-traitement mais également pour le réseau neuronal. Le principe de cette fonction n'a pas été modifié, mais sa mise en oeuvre l'a été légèrement. En effet, lors de la première soutenance, la formule que nous appliquions sur chqaue pixel était : $(rouge + bleu + vert)/3$. Or, depuis la première soutenance, nous avons modifié ces valeurs en effectuant une moyenne pondérée afin de nous adapter à la façons dont sont vues les couleurs sur un écran. Nous utilisons donc dorénavent cette formule : $0.3 * rouge + 0.59 * vert + 0.11 * bleu$.

2.4.3 Aie, j'ai mal aux yeux !

Cette étape du pré-traitement est importante car en une seule étape, c'est-à-dire avec un seul algorithme, nous corrigeons la luminosité de l'image et son contrast, mais c'est aussi l'occasion de lisser l'image, c'est-à-dire de diminuer les bruits parasites qui pourraient poser problème au réseau de neurones. Pour faire cela, nous utilisons un filtre et plusieurs passages de ce filtres dans différents sens. Nous avons déjà expliqué en profondeur cette étape dans notre premier rapport de soutenance, mais faisons un petit rappel ici, c'est important pour les étapes suivantes. Le premier passage est appelé `dilation`. Ce passage consiste à appliquer le maximum du filtre à un pixel, et ce pour chaque pixel de la surface.

Sur le même principe, mais en applicant le minimum du filtre à chaque pixel, nous avons ensuite appliqué une `erosion`.

Nous avons donc combiné ces deux transformations dans ce que l'on appelle une étape de `closing`. Cette étape consiste à appliquer une `dilation`, puis appliquer une `erosion` au résultat.

Le résultat du closing peut paraitre assez peu utilisable tel quel, et c'est en effet le cas. Il va falloir lui appliquer une dernière transformation. Cette dernière transformation peut paraitre un peu magique (et c'est le cas), consisite en la division de l'image de base et de l'image obtenue après l'étape du closing. Vous allez me dire, comment on divise deux images ? [3] Et bien c'est simple, il suffit de diviser leurs pixels entre eux un à un, et comme par magie, nous obtenons ça :

(a)

Avant correction

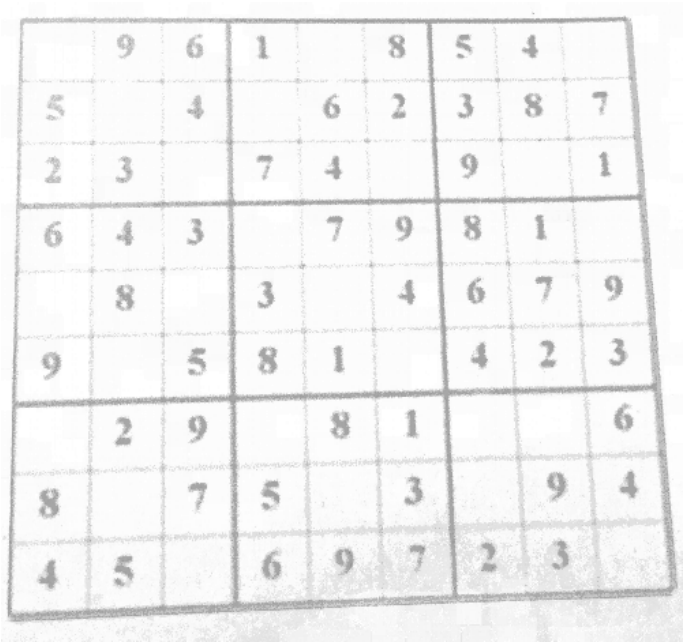
(b)

Après correction

Bien évidemment, tout cela n'a pas été simple. En fait, notre programme pour la première soutenance fonctionnait très bien sur les images où un certain contraste était déjà présent. En revanche, il fonctionnait très mal sur des images très peu contrastées comme l'image ci-dessous.

3. Vous ne direz probablement pas ça car nous ne devons pas être les premiers étudiants à avoir utilisé cette technique XD

Nous avons donc due jouer avec la taille et la forme des filtres utilisées pour avoir un résultat assez satisfaisant. Après de très nombreux essais, nous somme parvenus à ce résultat :



Voilà pour ce qui est de la partie sur la correction de la luminosité et du contraste et de la réduction de bruit. Nous espérons que ces comparaisons des images et ces explicitation des différentes étapes de cette partie vous aura permis de comprendre ce que nous avons voulu faire et comment cela fonctionne.

Passons maintenant à la binarisation des images, qui est la suite logique de la correction de contraste et de la réduction de bruit.

2.4.4 Il fait tout noir !

Vous l'aurez probablement compris, cette partie concerne la binarisation des images, c'est-à-dire la mise en noir et blanc des surfaces. Le problème de la fonction de binarisation que nous utilisions avant est que la valeure de seuil était constante, donc certaines images plus sombres que les autres n'étaient plus utilisables lors des étapes suivantes du prétraiment. Nous utilisons maintenant une méthode de binarisation avec un seuil adaptatif. L'idée de cette méthode est d'utiliser un seuil calculé à partir de la moyenne des pixels voisins. Ce seuil étant calculé sur un carré d'une taille faisant 1/8eme de l'image, l'algorithme naif utiliserait grossièrement, pour une image de dimensions NxN : $N^2 x (N/8)^2$ opérations. Ce qui correspond à plusieurs milliards d'opération pour une image de 1000x1000. Il existe heuresement des techniques pour réduire la complexité à N^2 (linéaire en nombre de pixels) en utilisant des images dites intégrales. Ces images peuvent être calculés en temps linéaire et permettent de trouver la moyenne des pixels voisins en temps constant. Cette technique est présentée dans l'article `Adaptive Thresholding Using Integral Image`[4]. Voici un comparatif entre l'ancienne et la nouvelle méthode :

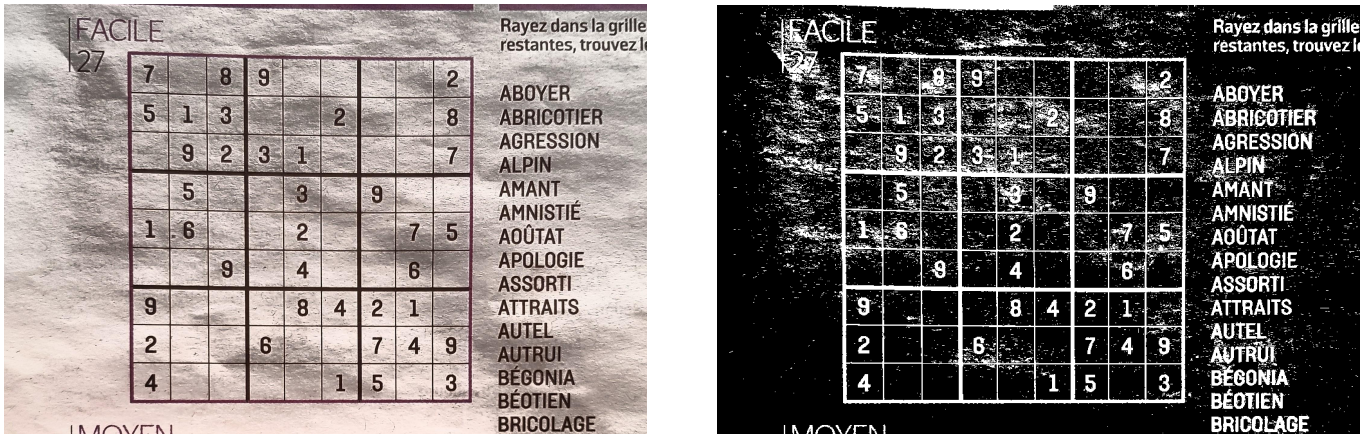


FIGURE 1 – Image originale, ancienne méthode

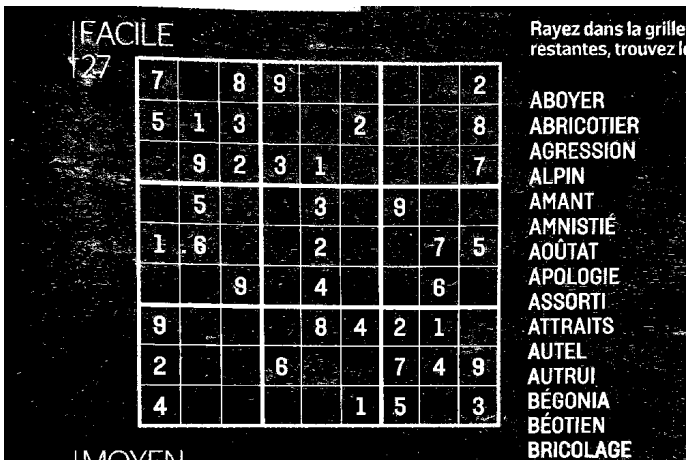
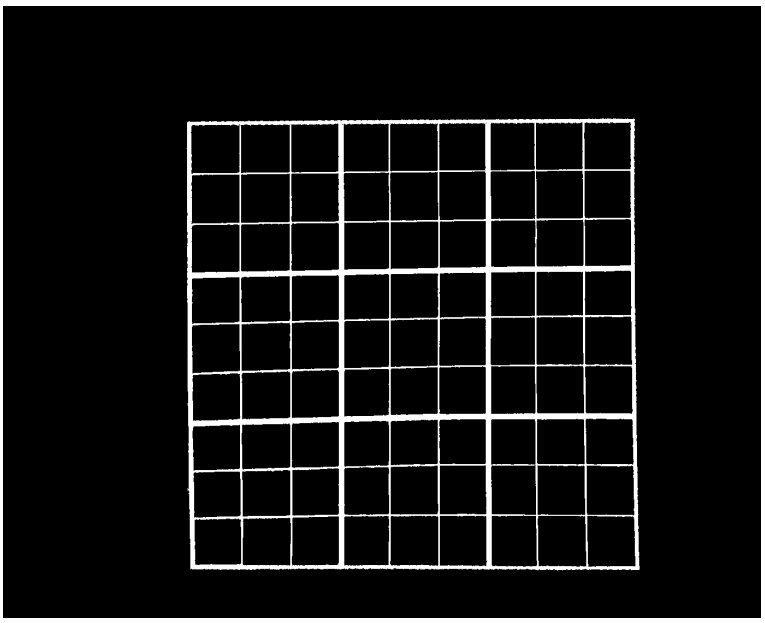


FIGURE 2 – Nouvelle méthode

4. Bradley, D., Roth, G. 2007. Adaptive Thresholding Using Integral Image. NRC 48816

2.4.5 Mais où est la grille ?

Pour trouver la grille c'est simple ! [5] On commence par diviser les images en différents sous-composants [6] grâce à un algorithme de remplissage par diffusion (flood-fill en anglais). Une fois une liste de composant établie, on ne garde que le composant qui a l'air le plus grand (le plus de pixels connectés entre eux). Notre image ressemble alors maintenant à ça :



De cette image, il ne nous reste plus qu'à chercher les pixels blancs aux extrémitées de la grille, ce qui est plutôt trivial.

2.4.6 Tournez les images !

Depuis la dernière soutenance, une étape importante du pré-traitement a été ajoutée : la rotation automatique.

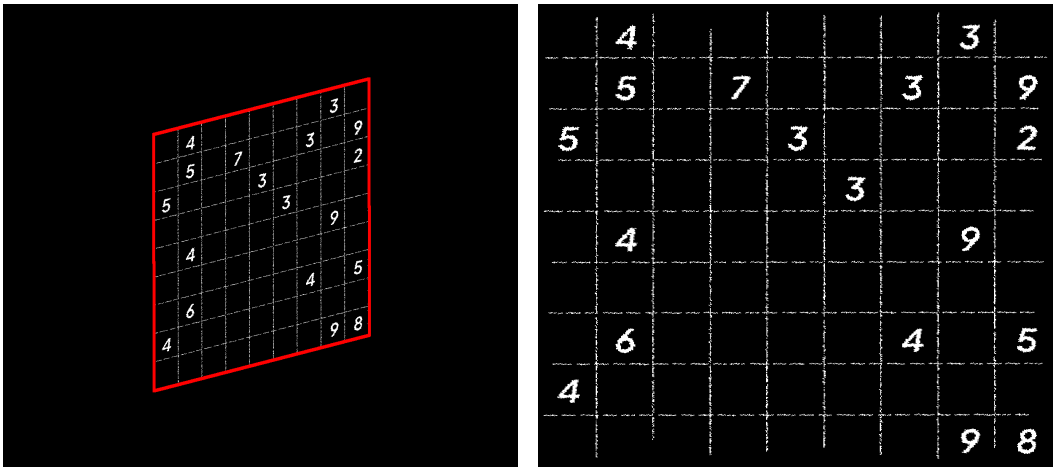
En effet, lors de la première soutenance, seule la rotation manuelle était disponible et fonctionnelle. Mais le passage de la rotation manuelle à la rotation automatique n'a pas été trop complexe. En effet, une fois la grille détectée, il nous suffit de faire un peu de mathématiques et de trigonométrie [7] pour trouver un angle approximatif.

Nous nous sommes contentés de cet angle approximatif puisque l'étape suivante va nous permettre d'affiner cette transformation avec une bien plus grande précision. C'est donc sur ces mots que nous vous laissons découvrir notre correction de perspective !

5. (non)
6. Une sous-composante d'une image est un ensemble de pixel tous reliés entre-eux
7. M. Ternier si vous passez par là c'est pour vous

2.4.7 Mais c'est pas droit cette histoire !

Grâce à une magnifique formule d'interpolation trouvée sur internet et à la détection de la grille effectuée à l'étape précédente, nous pouvons ainsi itérer sur chaque pixel de l'image pré-traitée et ainsi calculer la nouvelle position de chaque pixels sur la nouvelle image. Nous obtenons ainsi une image parfaitement droite et donc très facilement utilisable dans la suite du programme lors de la fin du pré-traitement et l'exécution du réseau de neurones.



(a) Avant transformation

(b) Après transformation

2.4.8 Découpons-les...

Cette partie du pré-traitement est la dernière étape avant la reconnaissance des chiffres par le réseau de neurones. C'est donc une étape importante puisque si les découpes ne sont pas correctes, les chiffres pourraient être coupé et donc inreconnaissables.

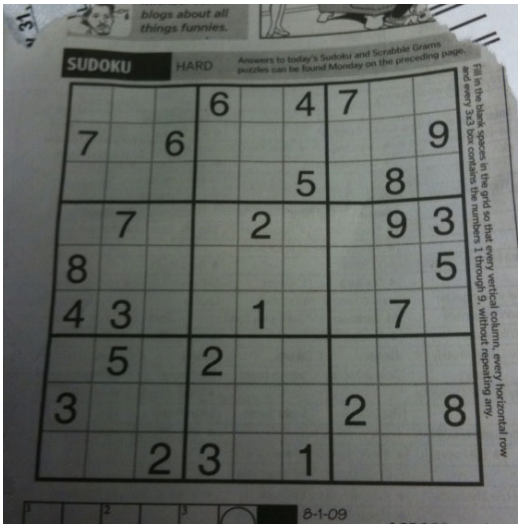
Heureusement, grâce au travail des différentes étapes précédentes du pré-traitement, nous obtenons un recadrage presque parfait, qui nous permet ainsi d'effectuer une simple division par 9 en largeur et en hauteur pour obtenir des résultats parfaitement utilisables par le réseau de neurones.

Voici un exemple de pré-traitement presque complet pour avoir une bonne idée du résultat et pouvoir faire la comparaison :

Voici un autre exemple qui permet de bien estimer le fonctionnement et les limites des algorithmes utilisés au cours des différentes étapes du pré-traitement :

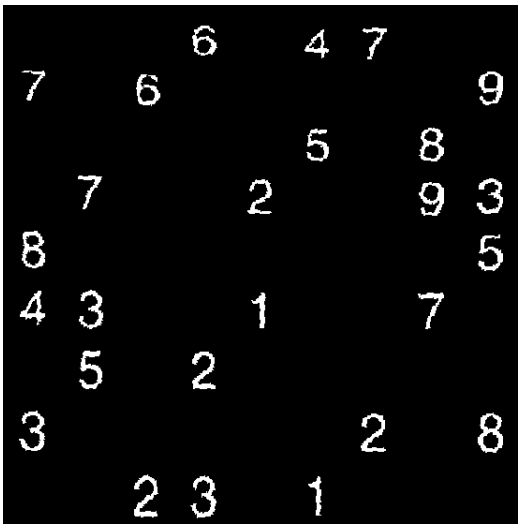
2.4.9 Sont-elles de la bonne taille ?

L'étape ultime avant d'envoyer toutes les données au réseau de neurones est une étape très simple et basique : elle consiste en un redimentionnement de chaque petite case pour que ces



(a)

Avant pré-traitement

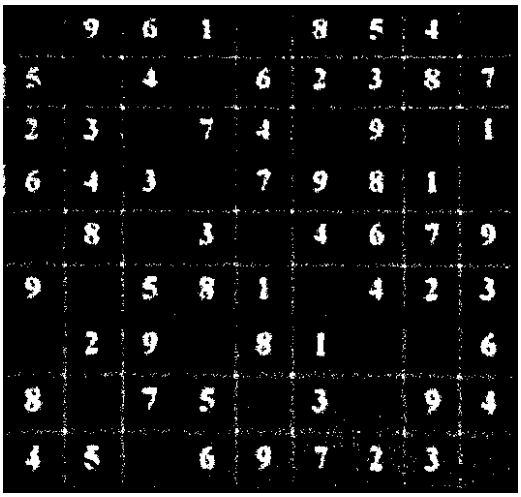


(b)

Après pré-traitement

(a)

Avant pré-traitement



(b)

Après pré-traitement

dernières fassent exactement 28 pixels de hauteur et 28 pixels de largeur. Cette étape, bien qu'elle ne soit pas indispensable, permet de simplifier et d'aider le réseau de neurones à sortir un résultat plus rapidement, nous permettant ainsi d'améliorer la rapidité de notre OCR.

2.4.10 De la couleur à la reconnaissance...

Contrairement à la première soutenance, le pré-traitement est maintenant entièrement terminé et fonctionnnel.

Passons maintenant à l'étape suivante de notre OCR : le réseau de neurones.

2.5 Le réseau de neurones

Pour créer le réseau de neurones qui pourra reconnaître les chiffres de nos grilles de sudokus, je me suis grandement inspiré des methodes du livre http://neuralnetworksanddeeplearning.com.

2.5.1 Optimisations

J'ai pu implémenter plusieurs optimisations pour l'apprentissage que conseille le livre :
- Normalisation des données
- Initialisation des poids avec une distribution gausienne
- Taux d'apprentissage dynamique (exponentiel)
- Fonction de cout d'entropie croisée

Voici brièvement à quoi correspondent ces optimisation :

La normalisation des données consiste à appliquer un fonction à l'ensemble des données avant que celles-ci ne soient utilisées par le réseau de neronnes, pour rendre le processus d'apprentissage plus rapide. La fonction de normalisation que nous avons utilisée s'appelle la variable centrée réduite. Ainsi, les données vues par le réseau de neronnes ont une moyenne de 0, et une variance de 1. Les données sont centrées autour de 0. La formule d'une telle transformation est la suivante : $f(x) = (x - \overline{X})/\sigma(X)$
$\overline{X}$ correspond à la moyenne de l'ensemble des données
$\sigma(X)$ correspond à l'écart type de l'ensemble des données

L'initialisation des poids avec une distribution gausienne signifie que les poids initiaux des réseaux de neronnes est assez similaire au point précédant dans le sens où le but est d'avoir des poids eux-aussi avec une moyenne egale à 0 et une variance de 1.
Le taux d'apprentissage exponentiel est une technique qui revient à multiplier le taux d'apprentissage par une valeur constante à chaque itération. Dans notre cas, cette constante est 1.03. Ainsi, au cours des 20 itérations, notre taux d'apprentissage passe de 0.7 à 1.3. Si nous métions notre taux d'apprentissage à 1.3 dès le début, le réseau de neronnes diverge et sa précision n'augmente pas.
La fonction de coût d'entropie croisée est plus adapté aux problèmes de classification. Contrairement à la fonction de coût que nous utilisions avant ; la fonction de coût quadratique.

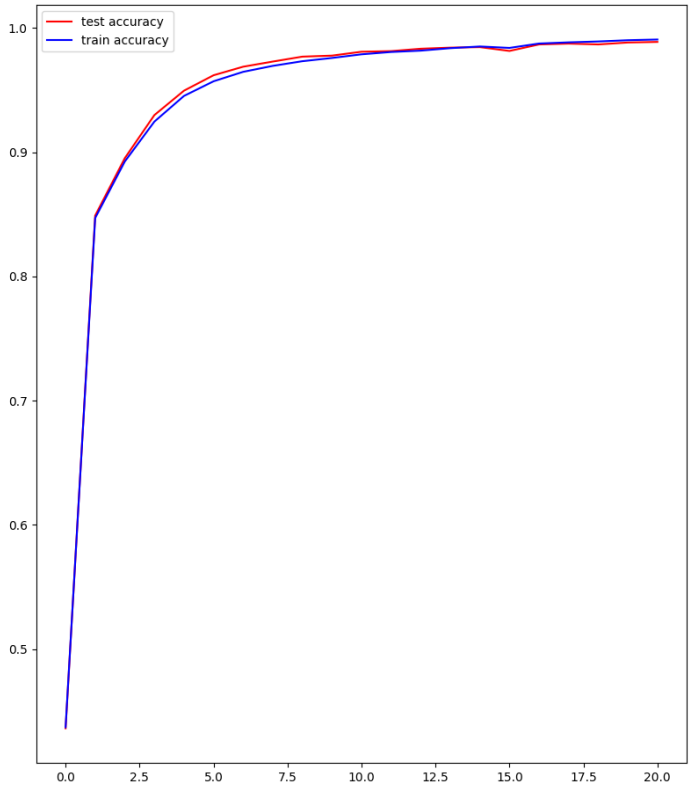
2.5.2 Résultats

Grâces aux différentes optimisations listées plus haut, en seulement 20 itérations, le réseau de neronnes est capable d'atteindre une précision de plus de 98% sur le jeu de données de test, ce qui est largement suffisant pour notre usage. En effet, 98% de précision ne veut pas dire qu'il n'a pas pu reconnâitre 2% des images, il a en fait reconnu toutes les images de tests. C'est la fonction de coût qui a eu en moyenne une valeure de 0.02.

```
➜  GOPY-OCR git:(oscar/adaptative-threshold) ✗ ./src/NeuralNetwork/main -p all
Running predictions on all images in data/testing/
|predict\expect|  0  |  1  |  2  |  3  |  4  |  5  |  6  |  7  |  8  |  9  |
|       0       |  37|     |     |     |     |     |     |     |     |     |
|       1       |     |  10|     |     |     |     |     |     |     |     |
|       2       |     |     |  10|     |     |     |     |     |     |     |
|       3       |     |     |     |  10|     |     |     |     |     |     |
|       4       |     |     |     |     |  10|     |     |     |     |     |
|       5       |     |     |     |     |     |  10|     |     |     |     |
|       6       |     |     |     |     |     |     |  11|     |     |     |
|       7       |     |     |     |     |     |     |     |  7|     |     |
|       8       |     |     |     |     |     |     |     |     |  8|     |
|       9       |     |     |     |     |     |     |     |     |     |  8|
| Total misses |    0|    0|    0|    0|    0|    0|    0|    0|    0|    0|
Grand total: 0/121 misses (100.00% accuracy)
```

De plus, nos itérations se font très rapidement car les "mini-batches" de données sont équitablement réparties sur un nombre de thread prédéfini, ce qui permets de quasiment multiplier la vitesse d'apprentissage par le nombre de coeurs du cpu de l'ordinateur. Pour mon cas, c'est 12 fois plus vite que l'apprentissage s'executait.
Même si une telle optimisation peut parraître inutile, car l'apprentissage n'a a se faire qu'une seule fois, de pouvoir constater de l'évolution de mes réseaux de neronnes aussi vite m'a permis d'essayer plus de combinaisons de paramètres plus vite, pour ainsi en arriver à pouvoir entraîner le réseau avec une telle précision en si peu d'itérations.

Voici un graphique représentant la précision du réseau de neronnes au cours de l'apprentissage :



2.5.3 Jeu de données

En ce qui concerne le jeu de données, nous pensions initialement utiliser MNist (Yann LeCun), mais lorsque la partie de prétraitement a été terminée, nous avons constaté que la précision n'était pas celle attendue.
J'ai alors décidé de créer un jeu de donnée customisé, adapté à notre prétraitement. Après avoir téléchargé un générateur de sudokus sur internet, j'en ai généré 200, je les ai prétraité et découpé en utilisant notre programme en ligne de commande. Ensuite, j'ai pu trier une première fois les images en utilisant notre ancien réseau de neronnes, et j'ai corrigé à la main les images mal classées. De plus, la première sortie de notre réseau de neronnes qui correspondait au chiffre 0 correspond désormais à une case vide (ou a des bouts de grilles).

2.6 La résolution

2.6.1 Et si nous chargions la grille ?

Cette étape n'a pas fondamentalement changé depuis la première soutenance. Malgré tout, une différence notable a été effectuée : la façon dont nous chargons la grille en mémoire. En effet, lors de la première soutenance, nous chargions la grille comme un tableau de 9 sous-tableaux. Maintenant, nous chargons la grille sous la forme d'un seul et unique tableau de longueur 81. Ce tableau est chargé en mémoire au format `row major`, ce qui nous permet d'accélérer le traitement en minimisant les déplacements au sein de la mémoire.

Comme vous allez le voir dans la suite, l'entièreté du solver a été adapté pour correspondre à ce comportement.

2.6.2 Résolvons-la !

Bien ! Maintenant que nous avons pu transformer l'image en une fichier texte puis une matrice, nous allons pouvoir entreprendre la phase de résolution de ce sudoku.

Résolution de sudoku ? Quelle coincidence ! L'année dernière nous avons justement eu la chance de travailler sur ce sujet en cours de programmation. Ainsi, nous avons pu aisément développer les différentes fonctions nécessaires à la résolution d'un sudoku. Nous avons utilisé un algorithme de `backtracking` car c'est le type d'algorithme le plus efficace à notre connaissance en ce qui concerne la résolution de grille de sudoku.
Parmi elles nous pouvons citer :

- `IsLineValid` (respectivement `IsColumnValid`) qui vérifie qu'une ligne (respectivement colonne) de la grille est valide
- `IsSquareValid` qui détermine si une grille de 3 par 3 est valide
- `IsBoardValid` qui renvoie si la board est valid ou non en utilisant donc les fonctions précédemment citées
- `IsSolved` qui determine si la grille est résolue
- Et enfin `Solve`, qui résout la grille de sudoku

Tout ce que nous avons dit précédemment datait de la première soutenance. Ce n'était qu'un rappel nécessaire pour la bonne compréhension de la suite de cette partie.

En effet, notre solveur fonctionnait, mais il n'était pas du tout optimisé. En effet, nous vérifions en fait que toute la grille était valide, et ce à chaque fois que nous essayions un chiffre à un emplacement donné. Malgré la vitesse de résolution parce que nous sommes en C, ce n'était pas du tout optimisé. Nous avons donc résolu ce problème depuis la première soutenance.

De plus, lors du premier rapport de soutenance, nous avions évoqués des tests unitaires. Ils étaient présents mais ne couvraient en aucun cas toutes les situations possible, et nous ne testions que la dernière fonction dans uniquement certaines situations.

Depuis la première soutenance, nous avons donc énormément amélioré ce système de tests unitaires puisque désormais, chaque fonction est testée de manière indépendante. De plus, nous avons drastiquement agrandie notre batterie de tests sur notre solver, ce qui nous a d'ailleurs permis de détecter certaines failles dans notre code.

Pour vous donner une idée de la façons dont sont faits nos tests unitaires, voici les échantillons de tests sur lesquels nous avons testé nos fonctions :

```
.
|-- not_solvable
|-- not_solved_not_valid_board
|-- not_solved_valid_board
|-- solvable
|-- solved_board
|-- valid_columns
|-- valid_lines
|-- valid_not_solved_board
|-- valid_solved_board
|-- valid_square
|-- wrong_board
|-- wrong_columns
|-- wrong_lines
`-- wrong_square

0 directories, 14 files
```

Nous avons donc estimé qu'avec ce nombre de tests unitaires, nos tests seraient exhaustifs sur tous les cas que ce dernier pourrait rencontrer lors de son exécution.

Maintenant que notre solver est complètement fonctionnel et optimisé, nous pouvons passer à l'explication des changements ayant eu lieu lors de l'étape suivante : l'enregistrement de la grille résolue.

2.6.3 Enregistrement en cours...

Comme mentionné précédemment, le fonctionnement du solver en entier a été migré vers une manipulation des données en `row major`, et de ce fait, la façon dont est enregistrée la grille aussi.

En fait, la migration de cette partie n'a pas été particulièrement compliqué. À vrai dire, cela nous a même permis d'optimiser notre solver puisque nous économisons un petit nombre d'opérations, même si quelques divisions et quelques modulo n'ont pas grand impact sur la complexité de ce type d'algorithme.

2.6.4 Ça y est, on a trouvé la solution !

Maintenant que nous avons trouvé la solution à notre sudoku, et après l'avoir enregistré dans un fichier texte, c'est la fin de cette étape de résolution. Il nous faut désormais afficher la résolution que nous avons trouvé, sinon tout ce que nous avons faits jusqu'ici n'aurait aucun intérêt.

Laissons donc place au `post-traitement`.

2.7 Le post-traitement

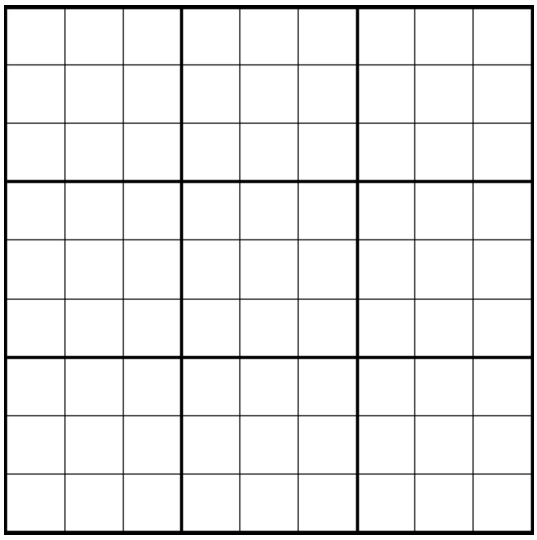
2.7.1 Récupération du résultat

Après tout ce travail, nous devions bien entendu représenter le résultat de notre OCR. Pour ce faire, nous avons choisi, non pas de remplir la grille prise en photo, mais d'en recréer une nouvelle. En effet, nous aurions pu réécrire par dessus la grille chargé au début de notre prétraitement, mais la complexité était assez importante puisque pour chaque image, il aurait fallut recalculer la position de la grille, sa taille, etc. Nous avons donc fait le choix de partir de rien pour recréer une grille résolue.

2.7.2 Création de la grille

Pour la création de cette grille, nous avions d'abord eu l'idée de la dessiner "en direct" en traçant des lignes plus ou moins épaisses avec les outils fournis par SDL2. Malheureusement, nous nous sommes vite rendu compte qu'en plus de rajouter du temps d'exécution pour pas grand chose, c'était aussi assez long et complexe pour obtenir un résultat agréable à voir. En effet, pour qu'un sudoku soit facilement lisible et compréhensible au premier coup d'oeil, il faut que les 9 grands carrés soient mis en valeur. La plupart du temps, cela consiste simplement en des traits plus épais autour de ces carrés, ainsi qu'un contour de la totalité du sudoku lui aussi plus épais.

Nous avons donc choisi de dessiner par dessus un modèle. Nous avons trouvé ce modèle sur internet, et nous l'avons adapté à notre utilisation. En effet, pour plus de simplicité de notre coté, nous avions besoin d'un modèle carré. Nous ne trouvions pas vraiment notre bonheur, alors nous avons pris un modèle rectangulaire que nous avons redimentionné et adapté à nos besoins avec GIMP. Voici le template que nous utilisons pour recréer la grille résolue :



2.7.3 Affichons la résolution

Maintenant que nous avons résolu le problème de la création de la grille en elle-même, il va nous falloir la remplir (oui, ça n'est pas très français mais faut pas le dire). Pour ce faire, nous chargeons la grille **avant** résolution, ainsi que la grille **après** résolution. L'intérêt de charger les deux grilles est que cela nous permet d'écrire d'une couleur différente les chiffres présents dans la grille de départ et les chiffres ayant été trouvés grâce à notre solver.

Nous remplissons donc la grille en créant des chiffres et les écrivant sur l'image de notre modèle. Nous avons choisi la police d'écriture `Archivo Narrow` pour écrire les chiffres, dans le but de garder une cohérence avec l'interface utilisateur.

Comme vous pouvez le voir sur l'image ci-dessous, nous écrivons en noir les chiffres déjà présent sur la grille de départ, et en bleu les chiffres ayant été trouvés lors de la résolution de la grille par notre solver.

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

Nous nous basons donc sur ce template pour écrire les chiffres par dessus et renvoyer une grille résolue.

Comme précédemment cité, nous faisons aussi en sorte de différencier les couleurs des chiffres détectés par le réseau de neurones et ceux trouvés par le solver. Comme vous pouvez le voir sur l'image suivante, voici à quoi ressemble une grille résolue en sortie de la fonction de post-traitement :

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

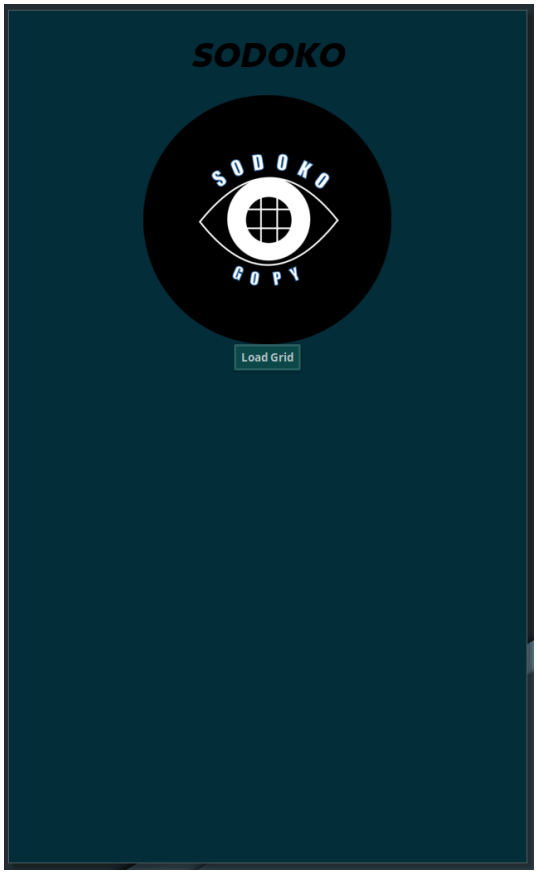
Nous en avons donc fini avec cette dernière et ultime partie de la résolution d'un sudoku via OCR. Mais ce n'est pas la fin pour autant. En effet, il est maintenant temps de vous présenter ce qui lie toutes les étapes de notre OCR et qui le rend réellement utilisable : l'interface graphique.

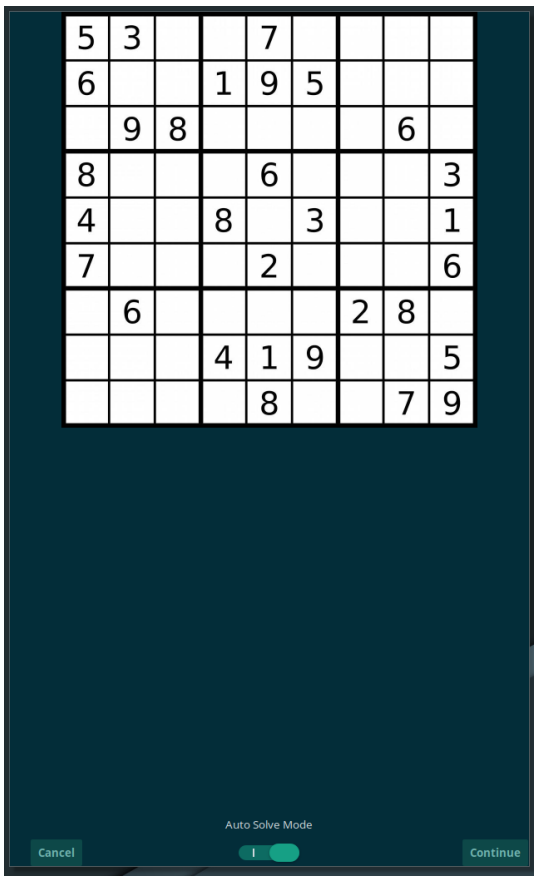
2.8 La GUI pour les petits

La GUI est un élément vital du projet si l'on veut que celui-ci puisse être utilisé par des profanes n'ayant pas sacrifié jour et nuit au dieu-machine. Cependant créer une interface en C à partir de rien est une tâche des plus complexes. C'est exactement pour cela que nous ne l'avons pas fait. A la place, la librairie GTK et l'outils Glade combinés nous ont sauvé la vie en permettant de gérer à assez haut niveau un grand nombre d'aspect de la GUI.

2.8.1 Fonctionnalités

La GUI devait respecter un certain nombre de points pour être considérée comme fonctionnelle.Tout d'abord, il fallait qu'elle soit claire et simple d'utilisation, ensuite il fallait qu'elle permette de charger une image choisie par l'utilisateur sur laquelle appliquer le programme. Finalement, il fallait que l'utilisateur puisse sauvegarder la grille de sudoku résolu. Cependant, parce que cela aurait été trop facile, et parce que nous n'aimons visiblement pas dormir, nous avons décidé d'implémenter quelques fonctionalités bonus. L'utilisateur peut en effet voir chaque étape de la résolution une à une et les sauvegarder individuellement. Cependant s'il n'est pas d'humeur à cliquer plein de fois sur le même bouton, il peut aussi choisir un mode de résolution automatique qui affichera une à une les étapes au fur et à mesure qu'elles sont complétées.



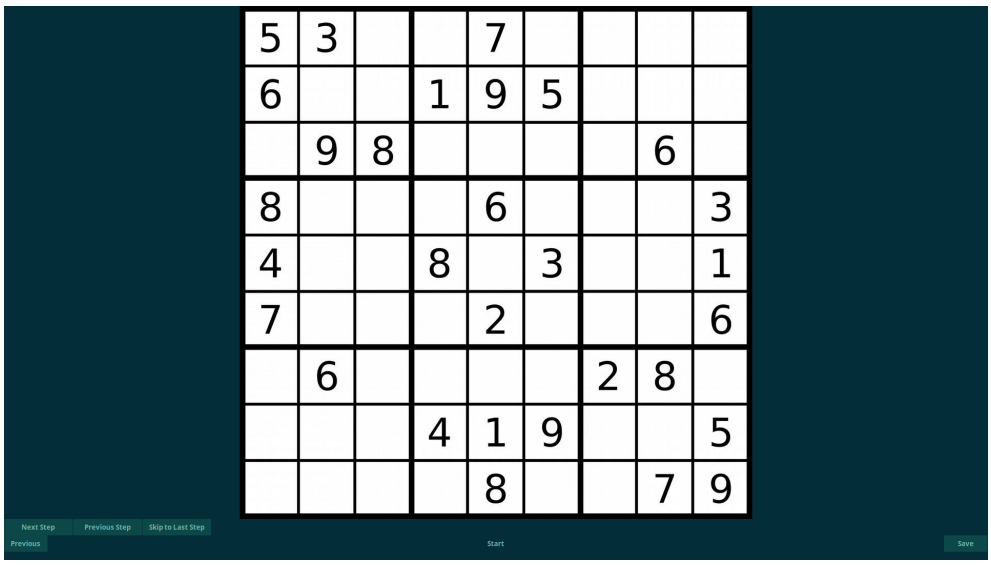


2.8.2 Une animation

Pour rendre notre interface graphique plus agréable à utiliser et plus fonctionnelle, nous avons décidé d'implémenter un mode manuel et un mode automatique. Dans le mode automatique comme manuel, toutes les étapes sont calculées et stockées dans une structure spéciale. C'est ensuite à l'entrée sur la page d'animation que les différences se sont entre le mode manuel et le mode automatique. Le principe du mode automatique est que la dernière étape (celle de la reconstruction de la grille résolue si le sudoku détecté par le réseau de neurones de l'intelligence artificielle est solvable, et la grille détectée par le réseau de neurones si cette dernière n'est pas solvable).

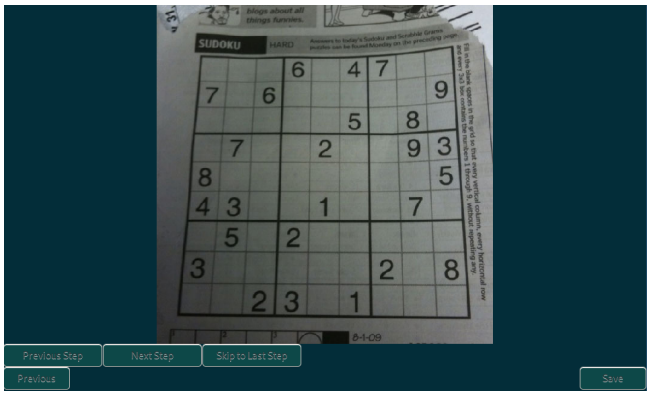
Ensuite, le mode manuel permet de partir de la première étape du pré-traitement, et d'exécuter les étapes une par une pour mieux comprendre le comportement de chacune.

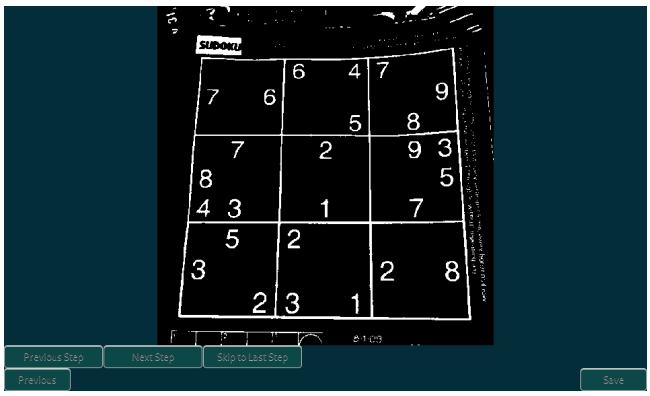
Bien entendu, nous avons aussi fait en sorte que, malgré le fait que le mode automatique soit activé, il est quand même possible d'accéder aux étapes précédentes en appuyant sur le bouton `Previous Step`.

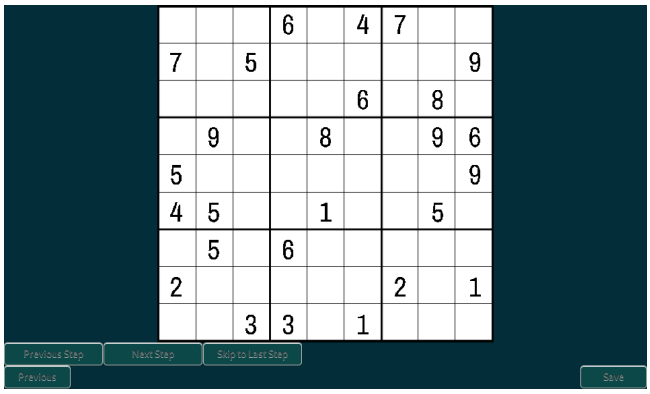


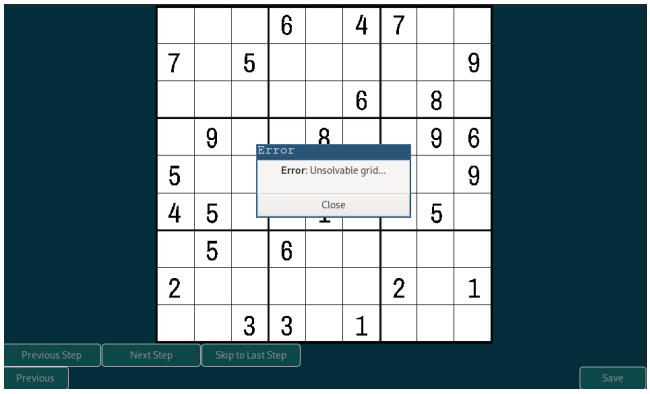
De plus, lorsque la grille n'est pas solvable, nous affichons une fenêtre d'erreur expliquant que le sudoku tel qu'il a été détecté par le réseau neuronale n'est pas solvable, et affiche ensuite la dernière étape s'étant correctement déroulée. Cette dernière étape correspond d'ailleurs toujours à la reconstruction de la grille détectée par notre intelligence artificielle.

Voici quelques images vous permettant de comprendre plus précisément le fonctionnement de notre animation de résolution et de fonctionnement de notre programme :





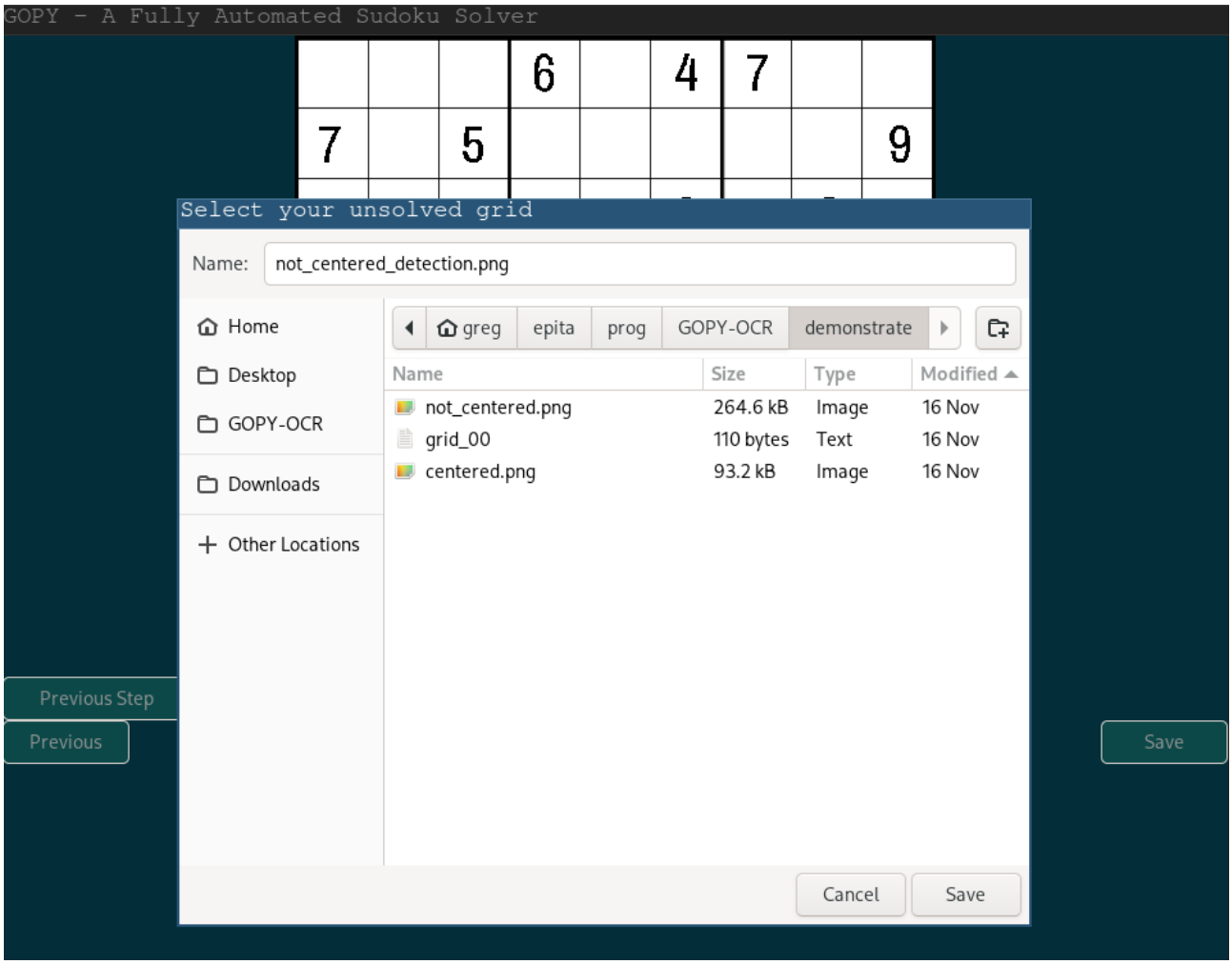




La dernière option de notre petite animation absoluement incroyablement magnifique (et je dis pas ça parce que c'est moi qui l'ai faite hein obviously), est la possibilité de sauvegarder chacune des étapes affichées par les boutons `Previous Step` et `Next Step`.

En effet, pour sauvergarder une image illustrant une des étapes du pré-traitement, du réseau de neurones ou bien même de la résolution du sudoku par notre solver, il vous suffit simplement, lorsque l'image est affiché sur l'écran, de cliquer sur le bouton `Save` afin qu'un explorateur de fichier s'affiche et que l'utilisateur puisse choisir le nom et l'emplacement où l'image correspondant à l'étape affichée sur l'écran va être sauvegardée.

Voici un petit exemple de l'affichage du choix de l'emplacement de sauvegarde. Par ailleurs, cet écran est très similaire à l'écran de choix du sudoku dont nous avons parlés un peu plus tôt dans cette section.




```
→  GOPY-OCR git:(main) ✗ ls demonstrate
centered.png  grid_00  not_centered.png  not_centered_detection.png
```

2.8.3 Des technalités...

Abordons maintenant la partie technique de la GUI, en effet la méthode d'affichage est un peu différente de ce que nous avons pu parfois voir. Nous n'avons pas créé de "GTK window" pour chaque page de l'application mais avons utilisé un système de pile afin de ne pas creer de pop-ups intempestifs et d'avoir de belles animations de transition.

Évidemment nous avons utilisé tout l'arsenal que glade mettait à notre disposition : des boutons, des leviers, des labels et bien plus, à la pelle pour obtenir un résultat non seulement fonctionnel mais également agréable à utiliser.

2.9 C'est déjà la fin...

Nous en avons donc terminé avec les parties de calcul pur. En effet, à partir de maintenant, nous avons toutes les fonctions et fonctionnalités formant notre OCR, mais nous n'avons pas un OCR pour autant. Il nous manque un liant à notre recette : l'interface utilisateur.

3 Des petits plus

Maintenant que nous avons présenté en détails les parties obligatoires de ce projet, nous allons présenter les parties bonus de notre programme, que nous avons codé dans le but de rendre ce projet encore meilleur et plus complet.

3.1 Un programme en ligne de commande

Un des bonus que nous avons implémentés est le fonctionnement en ligne de commande de notre programme. Ainsi, telle une véritable ligne de commande, nous pouvons utiliser et effectuer toutes les étapes de notre OCR, de la première étape du pré-trairement à la dernière étape du post-traitement sans avoir besoin d'accéder à une interface graphique.

Nous avions d'abord implémenté une version très basique lors de la première soutenance afin de nous permettre de facilement vous présenter les différentes fonctionnalités que nous avions implémentés. Nous l'avons ensuite amélioré pour intégrer les nouvelles fonctionnalités. Nous avons enfin implémenté les options permettant d'effectuer l'entièreté de notre OCR directement.

Voici une petite image permettant de voir quelles options nous avons choisi d'implémenter pour cette `CLI` (`Command Line Interface`, en opposition à la `GUI` pour `Graphical User Interface`) :

```
➜  GOPY-OCR git:(main) ./sodoko -h
Usage: demonstrate OPTION [FILE]
  -h,    --help                    Show this help
  -re,   --resize                  Resize the image to a 500x500 image to speed up the preprocess
  -g,    --grayscale IMG           Save the graysaled image in `IMG_grayscaled.png`
  -cb,   --contrast-brightness IMG Save the image with corrected contrast, brightness
                                   and noise reduced in `IMG_contrast.png`
  -b,    --binarization IMG        Save the binarized image in `IMG_bin.png`
  -r,    --rotate IMG ANGLE        Save the rotated image with the given angle
                                   in `IMG_ANGLE_rotated.png`
  -ar,   --auto-rotate IMG         Save the rotated image
                                   in `IMG_ANGLE_auto_rotated.png`
  -d,    --detect-grid IMG         Save the detected grid in `IMG_detected_grid.png`
  -pc,   --perspective-correct IMG Save the perspective correction of IMG
  -c,    --cut IMG                 Save the images in `IMG_X.png`
  -nn,   --neural-network IMG      Save the predicted grid in `IMG_detected_grid.sudoku`
  -S,    --solve GRID              Save the result in `GRID.result`
  -N,    --neural-network-poc      Show a proof of concept of the neural network
                                   Type `-N` to show help message about neural network
  -P,    --preprocess IMG          Execute the whole preprocess
  -O,    --postprocess GRID \     Generate an image of the solved grid
              SOLVED OUTPUT
  -A,    --all IMG                 Execute the whole process of the OCR.
                                   Save the detected grid in the same directory of IMG
                                   Returns 0 if everything is good
                                   Returns 1 if the detected grid is unsolvable
```

Et voici aussi un exemple de fonctionnement de toutes les étapes de notre programme, du pré-traitement à la détection des chiffres de la grille via le réseau de neurones :

```
➜  GOPY-OCR git:(main) ./sodoko -nn demonstrate/not_centered.png
Show full preprocess and neural network...
Sudoku:
... 6.4 7..
7.5 ... ..9
... ..6 .8.

.9. .8. .96
5.. ... ..9
45. .1. .5.

.5. 6.. ...
2.. ... 2.1
..3 3.1 ...

Saving to demonstrate/not_centered.sudoku...
```

Toutes les commandes sont sur le même principe, et nous permettent donc, en plus de pouvoir avoir un détail plus précis de chaque étape, de débugger plus facilement notre programme pour savoir quelle étape était dysfonctionnelle.

3.2 Un site internet

Parmi les bonus était proposée la conception d'un site internet. Nous avons ainsi choisis de le réaliser pour proposer un support à ce projet.

En ce qui concerne le style de ce site internet, nous avons choisi un theme sombre apportant un certaine modernité et simplicité, aux couleurs de notre logo. Les polices choisies sont simples pour une meilleure lisibilité.

Quant aux fonctionnalités, notre site internet vous propose une section "À PROPOS" à partir de laquelle vous découvrirez une courte description du projet ainsi que la présentation de chaque membre de cette belle équipe. Enfin, une section vous donne la possibilité de télécharger notre rapport pour entamer la lecture de cette aventure que nous avons vécu pendant ce court semestre.

Ci-dessous vous trouverez des images de chacunes des sections citées juste au-dessus.

Notre projet
Nous avons choisi, comme la première page le suggère, d'appeler notre équipe « GOPY », ce nom venant de la mise bout à bout des initiales de nos prénoms respectifs (Grégoire, Oscar, Pierre et Yness).
Nous avons aussi choisi d'appeler notre projet « Sodoko ». Il n'y a pas vraiment de raisons à ce nom, si ce n'est que le nom de « Suduku » nous paraissait un peu vulgraire mais que nous aimions bien la répétition d'une même voyelle dans le mot « sudoku ».

Téléchargez le rapport ici !
Rapport
Made with and

4 Notre ressentis sur le projet

4.1 Grégoire Lefaure

Je vais d'abord expliquer mon ressenti en tant que chef de projet sur le travail de groupe et la cohésion d'équipe durant la totalité de ce projet.

Ce projet a pour moi été très particulier. En effet, c'était la première fois que j'étais chef d'un projet d'équipe sur un aussi long terme, que ce soit un projet scolaire ou bien personnel. De plus, nous ne nous connaissions pas tous réellement avant le début du projet, ce qui n'a pas forcément aidé à atténuer les problèmes que je vais évoquer un peu plus tard.

Comme évoqué dans le premier rapport de soutenance, le travail de groupe était un peu complexe parfois, mais sans plus de problèmes que cela. En revanche, essentiellement sur la fin du projet, le stresse de l'arrivée de la deadline a pu créer quelques tensions au sein de l'équipe. En plus, le stresse de la deadline était accrue à cause du fait que nous n'avions pas, comme pour la première soutenance, de semaine sans aucun cours avant la récupération des projets par Epita. Nous avons donc tous dû trouver une organisation complexe, alliant ce projet et les différents TPs et DMs que nous devions effectuer en même temps. Et cette organisation n'a pas été facile à trouver, et l'organisation n'a d'ailleurs jamais été trouvée.

Ce manque d'organisation de notre part à d'ailleurs aussi été un peu la source de tensions dans notre groupe.

En effet, nous n'avons, objectivement, pas tous fournis le même travail pour cette seconde soutenance, et le non respect de certaines deadline à en effet pu créer quelques irritations car de nombreuses parties de ce projet sont interdépendantes et nous avons donc souvent besoin du code des autres membres de notre groupe pour pouvoir avancer. Ce constat est d'ailleurs d'autant plus véridique que notre mission principale, au-delà de finir toutes les fonctionnalités de manière indépendante, était de lier toutes ces fonctions ensemble dans le but de créer un programme complet et fonctionnel, et non pas de garder un programme composé de fonctionnalités indépendantes les unes des autres.

Certains autres problèmes que nous avons rencontré en tant que groupe était aussi de ne pas se marcher sur les pieds lorsque l'un trouvait que telle ou telle chose n'était pas assez bien faite à son goût. Il a fallut que tout le monde se mette d'accord pour ne pas se marcher dessus, même si ce n'était visiblement pas toujours facile à respecter.

Voilà, je crois que j'en ai fini avec mon ressenti sur le groupe en général et en tant que chef de projet, mais j'aimerais finir tout de même sur une note plus positive en expliquant mon ressenti plus personnel.

Concernant mon avis personnel sur ce projet, je dirais qu'il y a des choses très positives comme beaucoup moins. En effet, je n'ai pas trouvé ce projet extrêmement passionnant, bien que certaines parties soient intéressantes. Je me suis principalement occupé du pré-traitement, et j'ai trouvé que cette partie était un peu hasardeuse, on ne savait pas vraiment où on allait. Ça m'est d'ailleurs arrivé de modifier l'ordre d'appel de deux fonctions, et ainsi de passer de quelque chose qui n'était pas du tout fonctionnel à quelque chose de beaucoup mieux.

Au-delà de ça, il existe en fait des centaines d'algorithmes de traitement d'images, mais je n'ai pas trouvé qu'il y avait vraiment beaucoup de ressources qui nous indiquait le chemin, ou qui au moins nous montrait clairement les caractéristiques de chaque algorithme. En effet, pour de nombreuses parties du pré-traitement, il existe pléthore alrogithmes faisant tous plus ou moins la même chose. Et pourtant, choisir l'un par rapport à l'autre peut être un choix crucial pour la suite du processus.

C'est donc cette partie que j'ai le moins apprécié, le fait de naviguer à l'inconnu mais sans être dans l'inconnu pour autant. Malgré tout, ce projet m'a tout de même beaucoup apporté sur le plan des connaissances et des compétences techniques.

De plus, il y a bien évidemment certaines parties de ce projet que j'ai beaucoup aimé faire, comme par exemple la CLI. En effet, je trouve cela plus satisfaisant de faire un programme qui ressemble à une commande bash que d'appliquer des traitements sur une image pour la rendre exploitable par un réseau de neurones.

Pour conclure sur mon ressenti, je dirais que ce projet, bien qu'il ne soit pas et ne sera probablement jamais mon projet préféré, m'a tout de même beaucoup apporté.

4.2 Oscar Le Dauphin

Je me suis beaucoup investi dans ce projet, j'y ai passé du temps, de longues heures à débugger, écrire et réécrire du code, et j'ai énormément appris. Contrairement au projet du second semestre ou nous étions forcés d'utiliser un outil, Unity, que j'appréciais très peu, j'ai beaucoup apprécié développer en C sur celui ci. De plus, j'ai beaucoup apprécié la fluidité du développement que l'ont avait. Notament grâce aux systèmes de pull requests, de review, de CI/CD etc...

L'organisation du groupe a été, même si décrite comme problématique par les autres, ressentie comme étant un point positif de l'expérience. Contrairement au projet de l'année dernière, je passais plus de temps à coder qu'à resoudre des problèmes de conflits entre les fichiers.

Je conclurai en remerciant mon groupe sans qui un projet aussi abouti n'aurais jamais pu avoir lieu. Et merci pour cette expérience de travail en groupe.

4.3 Yness Khouader

La lecture commence à être longue mais voici à présent mon ressenti sur ce projet.

Je n'ai pas trouvé ce projet aussi amusant que celui du second semestre bien que je l'ai tout de même apprécié, probablement parce que nous avions une certaine liberté tandis que cette fois nous étions tous "contraints" de rendre le "même" projet.

À nouveau dans ce deuxième rapport j'estime que j'aurais pu / du fournir plus d'investissement et d'efforts pour mener au mieux ce projet.

J'ai pu découvrir que le développement d'une interface graphique n'était pas mon domaine de prédilection. En effet, j'ai très peu apprécié les outils que nous avons utilisé et jusqu'à aujourd'hui je ne les maîtrise pas.

À nouveau, dans un groupe de projet il y a des hauts et des bas. Je pourrais reprendre le terme utilisé par Grégoire plus tôt et évoquer certaines tensions quand la deadline s'approchait à grands pas.

Pour conclure, j'ai apprécié cette expérience qui fut enrichissante et j'ai hate de commencer un nouveau projet pour me developper tant dans mes compétences en informatique mais également dans mon acuité à travailler en groupe.

4.4 Pierre Wagniart

J'ai eu la chance de pouvoir beaucoup plus travailler dans cette dernière partie du projet. Ainsi j'ai pu me charger de tâche qui m'ont intéressées et permis d'évoluer en tant qu'ingénieur, notamment dans la gestion de la refactorisation d'un code, la compréhension du code de l'autre et la recherche de documentation.

J'ai aussi pu faire des erreurs dans plein de domaine du code à proprement parler et ainsi beaucoup progresser dans ces domaines. Coder la GUI a assurément rendu la tâche moins épuisante (tester une couleur de fond d'écran par erreur de pointeur permet de relativiser sur le débugage).

Par rapport à la première partie du projet où je me suis "occupé" d'une partie du prétraitement, la deuxième a été beaucoup plus intéressante et frustrante. De manière générale, le projet a eu pour moi des hauts comme des bas mais a été une experience humaine et technique des plus enrichissantes.

5 Conclusion

Voilà, nous sommes arrivés à la fin. Ce projet aura été, comme vous avez pu le constater au cours de ce rapport, une véritable épopée avec des hauts et des bas, mais nous sommes tout de même arrivés à produire un programme fonctionnel sur de nombreux cas.

Ce projet n'aura pas été facile, mais il nous a été très bénéfique sur de nombreux points que nous avons chacun pu détailler lors de l'expression de notre ressenti personnel.

Notre aventure se termine ici, et c'est en espérant que ce rapport de soutenance ait répondu à toutes vos attentes que nous nous quittons.

C'est la fin !