



## **Rapport de Projet**

Omri Zakaria DAKKA

Mathis PEROT

Fabio SINTONI

Paul FERREIRA

## Table des matières

<b>I - Introduction</b>	3
<b>II - Présentation du groupe</b>	4
1. Paul FERREIRA	4
2. Fabio SINTONI	5
3. Mathis PEROT	6
4. Omri Zakaria DAKKA	7
<b>III - Répartition des tâches</b>	8
<b>IV - Avancement du projet</b>	9
<b>V - Avancement individuel</b>	10
1. Paul FERREIRA	10
2. Fabio SINTONI et Mathis PEROT	15
3. Omri Zakaria DAKKA	22
<b>VI – Conclusion</b>	25
1. Paul FERREIRA	25
2. Fabio SINTONI	26
3. Mathis PEROT	27
4. Omri Zakaria DAKKA	28
5. Conclusion Générale	29

## **I. Introduction**

Dans le cadre de ce premier semestre de deuxième année à EPITA, il nous a été proposé de réaliser un projet de type OCR (Optical Character Recognition) visant à résoudre une grille de Sudoku à partir d'une image.

L'objectif final derrière ce projet est de nous offrir la possibilité de déployer toutes les connaissances que l'on a acquises durant notre parcours en plus de nous permettre d'esquisser une mise en situation face au monde réel, où il ne suffit plus de résoudre des fonctions, mais bien de devoir s'investir dans un travail de recherche laborieux comme ce sera le cas en entreprise.

Pour mettre en place un projet d'une telle envergure, nous avons décidé de le décomposer en quatre grandes parties qui sont : un programme de traitement de l'image et de découpage des caractères, un réseau de neurones permettant de reconnaître les chiffres dans la grille, un programme de résolution de la grille et enfin un logiciel combinant le tout, nous permettant ainsi le fonctionnement de notre OCR.

Ce rapport de soutenance contient la présentation de chaque membre du groupe, suivie de la répartition individuelle des tâches réalisées ainsi que le point culminant atteint dans la réalisation de ce projet avec la mise en avant des travaux et difficultés rencontrées par chacun.

## **II. Présentation du groupe**

### **1. Paul FERREIRA**

Je suis issu d'une terminale générale avec les spécialités mathématiques, numérique et sciences de l'informatique. J'ai par la suite intégré l'EPITA qui m'a permis de consolider mes connaissances informatiques et scientifiques. Pour ce projet, mon rôle est de m'occuper de la partie logiciel, rotation de l'image, création des résultats du programme ainsi que du solveur de sudoku. Je suis aussi le chef du groupe.

## 2. Fabio SINTONI

J'ai également eu mon baccalauréat général avec les spécialités mathématiques et physique-chimie. L'an passé, à EPITA, j'ai pu réaliser avec mon groupe un jeu vidéo en 2D en C# qui m'a permis de beaucoup progresser en programmation et notamment en programmation orientée objet. Cette année, je souhaite également m'améliorer et participer du mieux que je puisse à ce projet en réalisant un maximum de tâches. La partie qui m'a été confiée concernant le traitement d'images en C me paraît difficile car je ne m'y étais jamais essayé auparavant, mais j'ai essayé de réaliser au mieux la mission confiée par mon chef de projet.

### 3. Mathis PEROT

Je viens également d'une terminale générale avec les spécialités mathématique et NSI, en plus de l'option Mathématiques expertes suivie de la première année de classe préparatoire à EPITA où j'ai pu développer avec mon équipe un jeu en 2D de type boss rush en C# avec l'aide de l'outil Unity. Mon objectif pour cette année est d'en apprendre un maximum sur le traitement d'images, la programmation en C d'un logiciel ainsi que celle d'un réseau de neurones grâce à ce projet.

#### 4. Omri Zakaria DAKKA

Lors de mon parcours scolaire, je suis passé par une terminale générale avec spécialité mathématiques avant d'intégrer les classes préparatoires à l'EPITA qui m'ont permis de faire mes premiers pas dans le monde de l'informatique et de la programmation. Mon rôle au sein de ce projet est d'établir un réseau de neurones fonctionnel capable de reconnaître les chiffres dans une grille de Sudoku.

### **III. Répartition des tâches**

Nous avons convenu que deux des quatre parties de ce projet seraient à finir en premier à savoir le traitement de l'image et son découpage ainsi que le réseau de neurones. Nous avons donc créé deux binômes pour s'occuper de ces tâches majeures : deux personnes se sont donc totalement focalisées sur le traitement de l'image étant donné que c'est la partie demandant le plus de travail pour la première soutenance. Une autre personne s'est focalisée sur la réalisation du réseau de neurones avec une autre en soutien. Le logiciel et le solveur étant des tâches plus rapides à réaliser, ont été mis en place par la même personne.

<b>Tâches</b>	Omri Zakaria DAKKA	Mathis PEROT	Fabio SINTONI	Paul FERREIRA
Logiciel				<b>X</b>
Réseau de neurones	<b>X</b>			x
Traitement de l'image		<b>X</b>	<b>X</b>	
Résolveur de Sudoku				<b>X</b>

#### **Légende :**

**X** : tâche principale

x : tâche secondaire



#### **IV.   Avancement du projet**

Ci-dessous le tableau présentant l'avancement général du projet avec ce que nous avons prévu au départ et ce qui a été vraiment réalisé :

Tâches	Avancement réel
Prétraitement complet	70 %
Redressement manuel de l'image	100 %
Redressement automatique de l'image	0 %
Elimination des bruits parasites	100 %
Renforcement des contrastes	100 %
Réseau de neurones :	100 %
Apprentissage	100 %
Reconnaissance des chiffres de la grille	100 %
Reconstruction de la grille	70%
Résolution de la grille	100 %
Affichage de la grille	100 %
Sauvegarde du résultat	100 %
Interface graphique	100 %

## V. Avancements individuels

Dans cette partie chacun des membres du groupe va présenter ce qu'il a réalisé sur le projet.

### 1. Paul FERREIRA

J'ai en premiers lieu travaillé sur le logiciel permettant d'utiliser tous les éléments de notre projet. J'ai pour cela utilisé la bibliothèque libre de droit GTK qui simplifie grandement la création d'interface graphique en C et qui est multi-plateforme. J'ai couplé l'utilisation de cette bibliothèque au logiciel Glade qui permet de créer des interfaces graphiques en XML facilement pour GTK au format XML. Ce fichier est après converti par le Builder de GTK en un logiciel et je n'ai eu plus qu'à utiliser les signaux pour rendre le logiciel interactif.

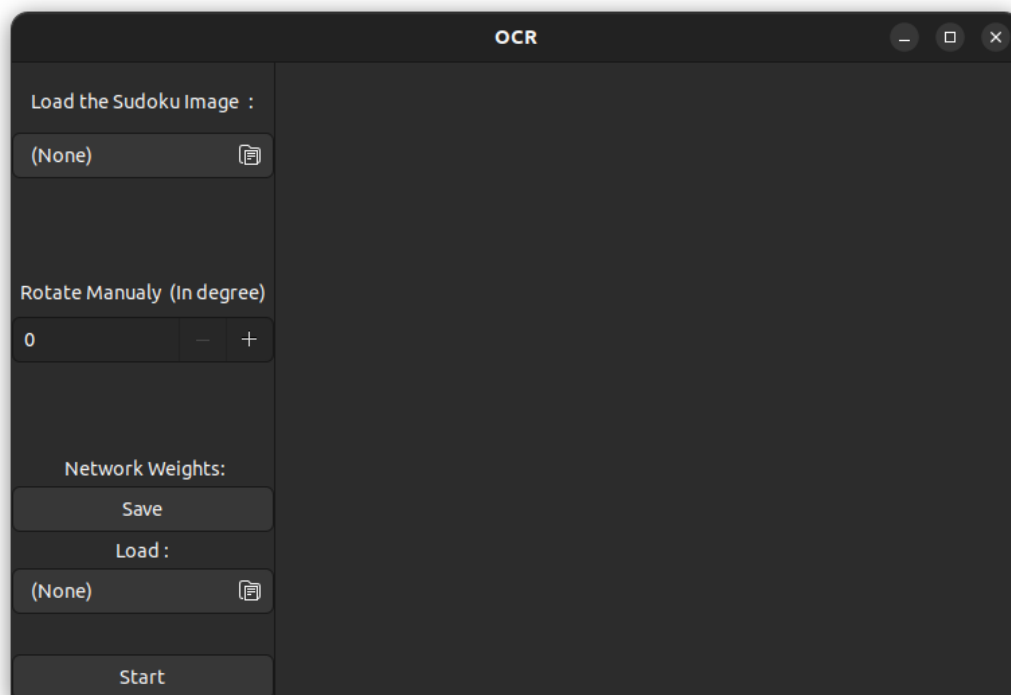


Figure 1 – Logiciel

J'ai implémenté par la suite le chargement d'une image dans le logiciel, pour cela j'ai utilisé l'objet Pixbuf dans GTK qui permet le contrôle de chacun des pixels de l'image ainsi que le redimensionnement de l'image afin qu'elle rentre dans la fenêtre.

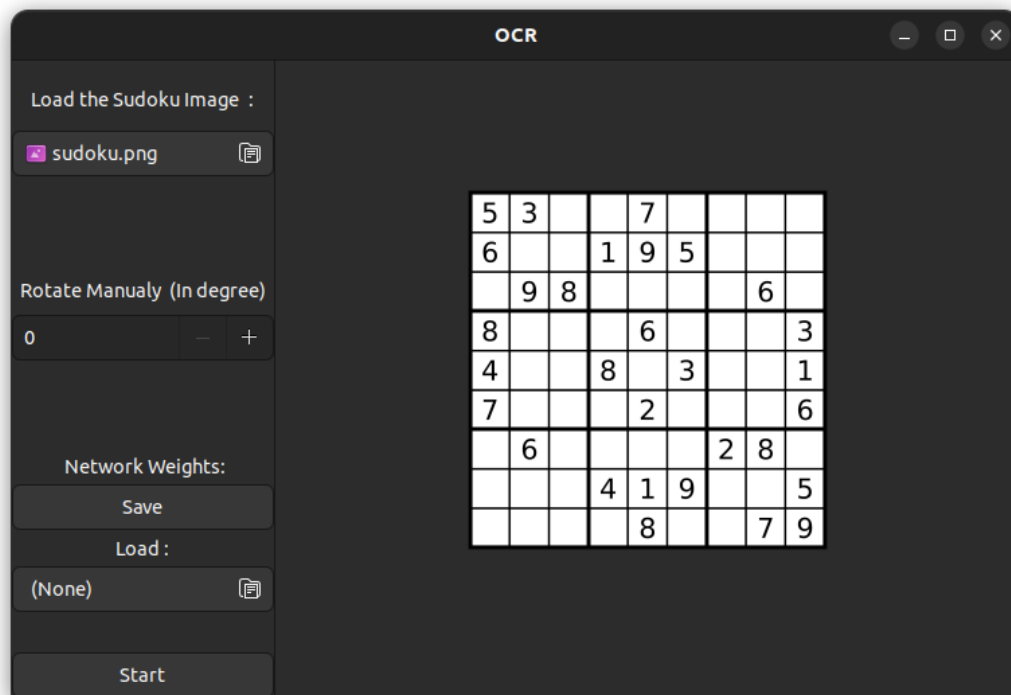


Figure 2 – Logiciel après chargement d'une image

Après avoir fait cela j'ai pu créer une fonction permettant la rotation de l'image avec un sélecteur. Pour cela, j'ai utilisé une formule assez simple de conversion de pixels, qui, bien qu'elle crée du grain fonctionne assez bien. Son principal problème résidant dans l'approximation des fonctions cosinus et sinus quand on converti des valeurs décimales en valeurs entières pour pouvoir rentrer dans une matrice de pixels.

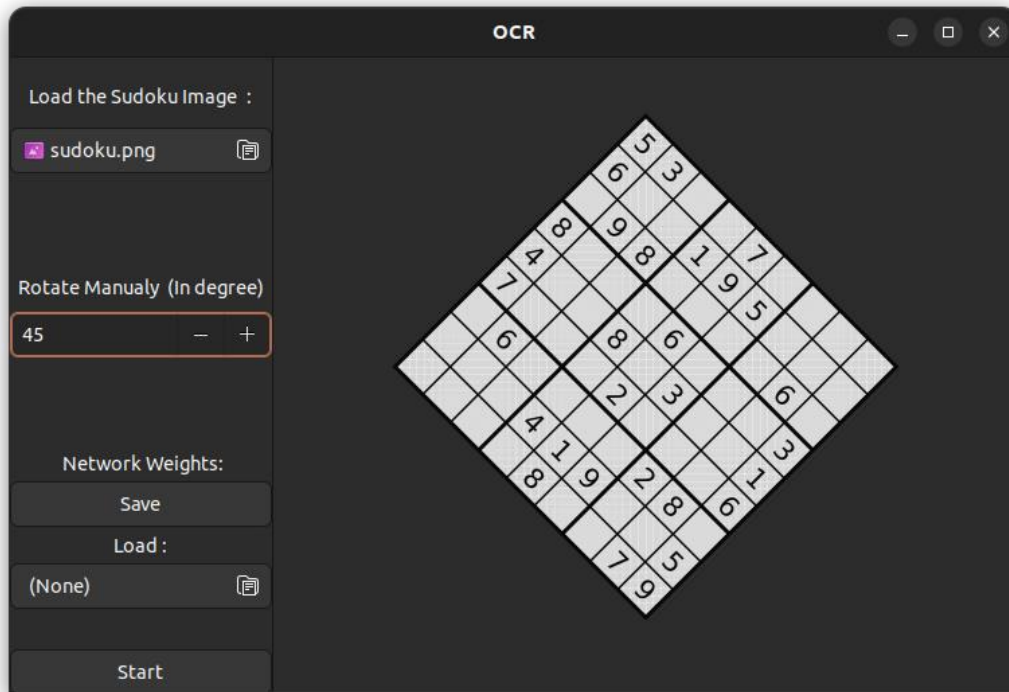


Figure 3 – Image après une rotation de 33°

Pour finir avec le logiciel j'ai aussi implémenté les fonctions permettant la lecture et l'enregistrement des poids du réseaux de neurones. Pour cela j'ai créé deux boutons : un bouton « Save » et un « FileChosserButton ». Le boutons « Save » va lancer l'apprentissage du réseau de neurones et écrire tous les poids simplement les uns après les autres dans un fichier nommé network.weights. Le bouton « Load » va lui s'occuper de lire les poids dans le fichier et les charger dans le réseau.

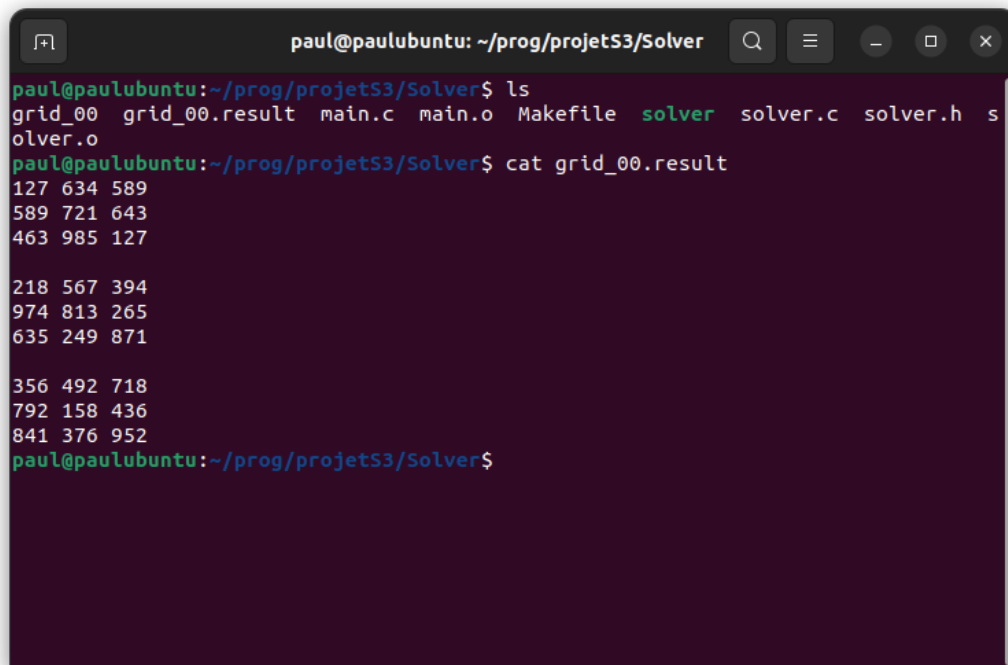
```

paul@paulbuntu:~/projet/projetS3/Software
0 -0.897103, 0.736916, 0.362680, -0.411966, 0.779984, 0.427163, 0.502384, -0.111904, 0.734385, -0.682083, -0.336816, -0.798250, -0.264625, -0.047833, 0.527489, 0.176533, 0.939633, -0.951486, 0.094775, -0.171878, 0.915910, -0.281311, -0.726619, -0.821674, -0.115793, -0.967081, -0.459045, -0.639263, 0.619839, 0.196483, -0.739487, -0.474663, -0.061401, 0.675463, 0.099961, -0.281417, 0.050436, -0.488212, 0.099279, -0.215179, -0.039925, -0.427537, -0.033432, 0.054549, -0.246420, -0.483249, -0.128012, 0.212992, -0.437426, 0.960727, -0.907885, -0.521516, -0.254554, -0.634264, -0.330653, -0.401554, 0.147482, 0.947782, -0.091725, -0.553922, -0.808219, -0.400787, 0.335087, 0.834254, 0.591796, -0.946330, -0.115310, -0.763445, 0.662949, 0.608511, 0.196039, -0.964588, -0.343918, -0.107920, 0.360042, 0.181422, 0.764063, -0.375965, 0.732716, 0.730820, -0.283850, -0.788800, -0.563735, 0.081546, -0.113590, 0.974082, 0.532272, -0.934585, 0.962898, 0.500957, -0.540697, -0.812713, -0.118817, 0.769591, -0.978458, 0.780751, 0.530661, -0.093768, 0.943306, 0.513609, -0.424257, 0.139336, 0.549021, 0.231825, -0.367884, -0.090937, -0.590823, 0.790029, 0.530097, -0.065318, 0.526898, -0.150153, -0.454117, 0.013354, 0.330893, 0.218012, -0.180825, 0.232768, 0.988203, -0.080869, 0.742111, -0.047324, -0.640899, 0.538203, -0.883763, -0.341448, 0.388802, 0.080450, -0.900942, -0.097528, 0.656193, 0.149094, -0.548057, -0.111982, 0.181111, 0.360555, 0.289985, -0.028811, -0.106347, 0.411072, -0.443722, 0.195090, 0.890331, -0.480558, -0.539203, -0.011333, -0.494076, 0.336558, -0.736258, -0.572877, 0.598708, -0.958339, -0.425001, 0.955802, 0.543102, -0.251384, -0.978346, -0.067676, 0.829806, -0.969189, 0.834496, 0.485259, 0.179906, -0.714012, -0.626723, -0.338984, 0.646544, 0.663601, 0.338206, -0.524989, 0.827743, 0.777188, 0.683096, -0.141522, -0.680455, -0.843111, 0.842820, -0.234121, 0.441658, -0.833338, 0.147602, 0.838358, -0.088369, 0.762801, -0.885449, 0.654793, -0.488303, 0.616511, -0.433183, -0.603177, 0.477418, -0.221488, 0.064359, 0.283303, -0.143806, 0.794376, -0.063891, 0.609372, 0.574860, -0.306416, 0.730448, 0.188101, -0.837310, 0.523717, 0.390301, -0.918411, 0.827487, -0.379683, -0.802806, -0.520221, -0.108404, -0.270668, -0.060997, 0.388125, 0.066660, 0.605624, -0.527168, 0.561835, -0.866365, 0.483396, -0.950954, 0.610553, -0.738182, 0.113397, -0.183643, 0.117920, -0.088227, 0.832466, -0.114167, -0.473958, -0.451863, -0.928686, 0.088740, -0.924817, 0.470237, 0.175329, 0.499260, -0.909447, 0.137263, -0.425991, -0.018491, -0.080835, 0.109112, -0.038368, -0.929374, -0.085244, -0.157526, 0.832459, 0.048391, -0.074220, 0.681586, -0.340556, -0.417482, -0.205088, 0.535881, 0.705519, 0.706075, 0.388266, -0.470089, -0.407402, 0.943316, 0.871148, -0.333558, 0.083837, 0.143881, -0.863322, -0.821654, 0.842261, -0.727268, 0.535183, 0.412380, -0.841741, 0.657985, -0.2178568, 0.578375, 0.723611, 0.636188, -0.579152, 0.356078, -0.315421, -0.253372, 0.037575, 0.344024, 0.334220, 0.832478, -0.108175, 0.039745, 0.539153, -0.711909, 0.502756, -0.868340, -0.797592, -0.360696, -0.201898, 0.205464, 0.776905, -0.685228, 0.883658, -0.180834, 0.162012, -0.065131, -0.768334, -0.029248, -0.412146, -0.047182, 0.049127, -0.088336, -0.410914, -0.038823, 0.607534, 0.273665, 0.279485, 0.474797, -0.396362, 1.282779, -0.545240, -0.748454, 0.111838, -0.005159, -0.931211, 0.156083, -0.284891, -0.382311, -0.549170, -0.462413, 0.517524, 0.496572, -0.923263, 0.885685, 0.053331, -0.791608, -0.919188, 0.687234, 0.060501, 0.213477, 0.464139, 0.841581, -0.402074, -0.603177, 0.844625, -0.578688, 0.829491, -0.737476, -0.292699, -0.801782, -0.412454, -0.646156, 0.861479, 0.925752, -0.105599, -0.058593, 0.768045, -0.422863, 0.807885, -0.922410, -0.265974, 0.650806, -0.155539, -0.824316, 0.757475, 0.968928, 0.214042, 0.689106, 0.730373, -0.791799, 0.523899, -0.496318, 0.224014, -0.889285, -0.156628, -0.729361, -0.467972, -0.531686, -0.502831, 0.239328, -0.333468, 0.084715, 0.593173, -0.471989, 0.010467, 0.512707, 0.277418, -0.221488, 0.064359, 0.283303, -0.143806, 0.794376, -0.063891, 0.609372, 0.574860, -0.306416, 0.730448, 0.188101, -0.837310, 0.523717, 0.390301, -0.918411, 0.827487, -0.379683, -0.802806, -0.520221, -0.108404, -0.270668, -0.060997, 0.388125, 0.066660, 0.605624, -0.527168, 0.561835, -0.866365, 0.483396, -0.950954, 0.610553, -0.738182, 0.113397, -0.183643, 0.117920, -0.088227, 0.832466, -0.114167, -0.473958, -0.451863, -0.928686, 0.088740, -0.924817, 0.470237, 0.175329, 0.499260, -0.909447, 0.137263, -0.425991, -0.018491, -0.080835, 0.109112, -0.038368, -0.929374, -0.085244, -0.157526, 0.832459, 0.048391, -0.074220, 0.681586, -0.340556, -0.417482, -0.205088, 0.535881, 0.705519, 0.706075, 0.388266, -0.470089, -0.407402, 0.943316, 0.871148, -0.333558, 0.083837, 0.143881, -0.863322, -0.821654, 0.842261, -0.727268, 0.535183, 0.412380, -0.841741, 0.657985, -0.2178568, 0.578375, 0.723611, 0.636188, -0.579152, 0.356078, -0.315421, -0.253372, 0.037575, 0.344024, 0.334220, 0.832478, -0.108175, 0.039745, 0.539153, -0.711909, 0.502756, -0.868340, -0.797592, -0.360696, -0.201898, 0.205464, 0.776905, -0.685228, 0.883658, -0.180834, 0.162012, -0.065131, -0.768334, -0.029248, -0.412146, -0.047182, 0.049127, -0.088336, -0.410914, -0.038823, 0.607534, 0.273665, 0.279485, 0.474797, -0.396362, 1.282779, -0.545240, -0.748454, 0.111838, -0.005159, -0.931211, 0.156083, -0.284891, -0.382311, -0.549170, -0.462413, 0.517524, 0.496572, -0.923263, 0.885685, 0.053331, -0.791608, -0.919188, 0.687234, 0.060501, 0.213477, 0.464139, 0.841581, -0.402074, -0.603177, 0.844625, -0.578688, 0.829491, -0.737476, -0.292699, -0.801782, -0.412454, -0.646156, 0.861479, 0.925752, -0.105599, -0.058593, 0.768045, -0.422863, 0.807885, -0.922410, -0.265974, 0.650806, -0.155539, -0.824316, 0.757475, 0.968928, 0.214042, 0.689106, 0.730373, -0.791799, 0.523899, -0.496318, 0.224014, -0.889285, -0.156628, -0.729361, -0.467972, -0.531686, -0.502831, 0.239328, -0.333468, 0.084715, 0.593173, -0.471989, 0.010467, 0.512707, 0.277418, -0.221488, 0.064359, 0.283303, -0.143806, 0.794376, -0.063891, 0.609372, 0.574860, -0.306416, 0.730448, 0.188101, -0.837310, 0.523717, 0.390301, -0.918411, 0.827487, -0.379683, -0.802806, -0.520221, -0.108404, -0.270668, -0.060997, 0.388125, 0.066660, 0.605624, -0.527168, 0.561835, -0.866365, 0.483396, -0.950954, 0.610553, -0.738182, 0.113397, -0.183643, 0.117920, -0.088227, 0.832466, -0.114167, -0.473958, -0.451863, -0.928686, 0.088740, -0.924817, 0.470237, 0.175329, 0.499260, -0.909447, 0.137263, -0.425991, -0.018491, -0.080835, 0.109112, -0.038368, -0.929374, -0.085244, -0.157526, 0.832459, 0.048391, -0.074220, 0.681586, -0.340556, -0.417482, -0.205088, 0.535881, 0.705519, 0.706075, 0.388266, -0.470089, -0.407402, 0.943316, 0.871148, -0.333558, 0.083837, 0.143881, -0.863322, -0.821654, 0.842261, -0.727268, 0.535183, 0.412380, -0.841741, 0.657985, -0.2178568, 0.578375, 0.723611, 0.636188, -0.579152, 0.356078, -0.315421, -0.253372, 0.037575, 0.344024, 0.334220, 0.832478, -0.108175, 0.039745, 0.539153, -0.711909, 0.502756, -0.868340, -0.797592, -0.360696, -0.201898, 0.205464, 0.776905, -0.685228, 0.883658, -0.180834, 0.162012, -0.065131, -0.768334, -0.029248, -0.412146, -0.047182, 0.049127, -0.088336, -0.410914, -0.038823, 0.607534, 0.273665, 0.279485, 0.474797, -0.396362, 1.282779, -0.545240, -0.748454, 0.111838, -0.005159, -0.931211, 0.156083, -0.284891, -0.382311, -0.549170, -0.462413, 0.517524, 0.496572, -0.923263, 0.885685, 0.053331, -0.791608, -0.919188, 0.687234, 0.060501, 0.213477, 0.464139, 0.841581, -0.402074, -0.603177, 0.844625, -0.578688, 0.829491, -0.737476, -0.292699, -0.801782, -0.412454, -0.646156, 0.861479, 0.925752, -0.105599, -0.058593, 0.768045, -0.422863, 0.807885, -0.922410, -0.265974, 0.650806, -0.155539, -0.824316, 0.757475, 0.968928, 0.214042, 0.689106, 0.730373, -0.791799, 0.523899, -0.496318, 0.224014, -0.889285, -0.156628, -0.729361, -0.467972, -0.531686, -0.502831, 0.239328, -0.333468, 0.084715, 0.593173, -0.471989, 0.010467, 0.512707, 0.277418, -0.221488, 0.064359, 0.283303, -0.143806, 0.794376, -0.063891, 0.609372, 0.574860, -0.306416, 0.730448, 0.188101, -0.837310, 0.523717, 0.390301, -0.918411, 0.827487, -0.379683, -0.802806, -0.520221, -0.108404, -0.270668, -0.060997, 0.388125, 0.066660, 0.605624, -0.527168, 0.561835, -0.866365, 0.483396, -0.950954, 0.610553, -0.738182, 0.113397, -0.183643, 0.117920, -0.088227, 0.832466, -0.114167, -0.473958, -0.451863, -0.928686, 0.088740, -0.924817, 0.470237, 0.175329, 0.499260, -0.909447, 0.137263, -0.425991, -0.018491, -0.080835, 0.109112, -0.038368, -0.929374, -0.085244, -0.157526, 0.832459, 0.048391, -0.074220, 0.681586, -0.340556, -0.417482, -0.205088, 0.535881, 0.705519, 0.706075, 0.388266, -0.470089, -0.407402, 0.943316, 0.871148, -0.333558, 0.083837, 0.143881, -0.863322, -0.821654, 0.842261, -0.727268, 0.535183, 0.412380, -0.841741, 0.657985, -0.2178568, 0.578375, 0.723611, 0.636188, -0.579152, 0.356078, -0.315421, -0.253372, 0.037575, 0.344024, 0.334220, 0.832478, -0.108175, 0.039745, 0.539153, -0.711909, 0.502756, -0.868340, -0.797592, -0.360696, -0.201898, 0.205464, 0.776905, -0.685228, 0.883658, -0.180834, 0.162012, -0.065131, -0.768334, -0.029248, -0.412146, -0.047182, 0.049127, -0.088336, -0.410914, -0.038823, 0.607534, 0.273665, 0.279485, 0.474797, -0.396362, 1.282779, -0.545240, -0.748454, 0.111838, -0.005159, -0.931211, 0.156083, -0.284891, -0.382311, -0.549170, -0.462413, 0.517524, 0.496572, -0.923263, 0.885685, 0.053331, -0.791608, -0.919188, 0.687234, 0.060501, 0.213477, 0.464139, 0.841581, -0.402074, -0.603177, 0.844625, -0.578688, 0.829491, -0.737476, -0.292699, -0.801782, -0.412454, -0.646156, 0.861479, 0.925752, -0.105599, -0.058593, 0.768045, -0.422863, 0.807885, -0.922410, -0.265974, 0.650806, -0.155539, -0.824316, 0.757475, 0.968928, 0.214042, 0.689106, 0.730373, -0.791799, 0.523899, -0.496318, 0.224014, -0.889285, -0.156628, -0.729361, -0.467972, -0.531686, -0.502831, 0.239328, -0.333468, 0.084715, 0.593173, -0.471989, 0.010467, 0.512707, 0.277418, -0.221488, 0.064359, 0.283303, -0.143806, 0.794376, -0.063891, 0.609372, 0.574860, -0.306416, 0.730448, 0.188101, -0.837310, 0.523717, 0.390301, -0.918411, 0.827487, -0.379683, -0.802806, -0.520221, -0.108404, -0.270668, -0.060997, 0.388125, 0.066660, 0.605624, -0.527168, 0.561835, -0.866365, 0.483396, -0.950954, 0.610553, -0.738182, 0.113397, -0.183643, 0.117920, -0.088227, 0.832466, -0.114167, -0.473958, -0.451863, -0.928686, 0.088740, -0.924817, 0.470237, 0.175329, 0.499260, -0.909447, 0.137263, -0.425991, -0.018491, -0.080835, 0.109112, -0.038368, -0.929374, -0.085244, -0.157526, 0.832459, 0.048391, -0.074220, 0.681586, -0.340556, -0.417482, -0.205088, 0.535881, 0.705519, 0.706075, 0.388266, -0.470089, -0.407402, 0.943316, 0.871148, -0.333558, 0.083837, 0.143881, -0.863322, -0.821654, 0.842261, -0.727268, 0.535183, 0.412380, -0.841741, 0.657985, -0.2178568, 0.578375, 0.723611, 0.636188, -0.579152, 0.356078, -0.315421, -0.253372, 0.037575, 0.344024, 0.334220, 0.832478, -0.108175, 0.039745, 0.539153, -0.711909, 0.502756, -0.868340, -0.797592, -0.360696, -0.201898, 0.205464, 0.776905, -0.685228, 0.883658, -0.180834, 0.162012, -0.065131, -0.768334, -0.029248, -0.412146, -0.047182, 0.049127, -0.088336, -0.410914, -0.038823, 0.607534, 0.273665, 0.279485, 0.474797, -0.396362, 1.282779, -0.545240, -0.748454, 0.111838, -0.005159, -0.931211, 0.156083, -0.284891, -0.382311, -0.549170, -0.462413, 0.517524, 0.496572, -0.923263, 0.885685, 0.053331, -0.791608, -0.919188, 0.687234, 0.060501, 0.213477, 0.464139, 0.841581, -0.402074, -0.603177, 0.844625, -0.578688, 0.829491, -0.737476, -0.292699, -0.801782, -0.412454, -0.646156, 0.861479, 0.925752, -0.105599, -0.058593, 0.768045, -0.422863, 0.807885, -0.922410, -0.265974, 0.650806, -0.155539, -0.824316, 0.757475, 0.968928, 0.214042, 0.689106, 0.730373, -0.791799, 0.523899, -0.496318, 0.224014, -0.889285, -0.156628, -0.729361, -0.467972, -0.531686, -0.502831, 0.239328, -0.333468, 0.084715, 0.593173, -0.471989, 0.010467, 0.512707, 0.277418, -0.221488, 0.064359, 0.283303, -0.143806, 0.794376, -0.063891, 0.609372, 0.574860, -0.306416, 0.730448, 0.188101, -0.837310, 0.523717, 0.390301, -0.918411, 0.827487, -0.379683, -0.802806, -0.520221, -0.108404, -0.270668, -0.060997, 0.388125, 0.066660, 0.605624, -0.527168, 0.561835, -0.866365, 0.483396, -0.950954, 0.610553, -0.738182, 0.113397, -0.183643, 0.117920, -0.088227, 0.832466, -0.114167, -0.473958, -0.451863, -0.928686, 0.088740, -0.924817, 0.470237, 0.175329, 0.499260, -0.909447, 0.137263, -0.425991, -
```

Figure 4 – Fichier « network.weights »

Je me suis par la suite concentré sur le solveur de sudoku. Je l'ai réalisé grâce un algorithme récursif qui va placer un chiffre de 1 à 9 dans la grille et qui va vérifier si ce nombre ne rentre pas en collision avec des nombres placées précédemment ou déjà présents dans la grille. Si aucun des 9 chiffres ne rentre dans la grille, l'algorithme revient en arrière et va choisir d'autres chiffres.

Voici le résultat du solveur à la fin de l'exécution :



```
paul@paulubuntu: ~/prog/projetS3/Solver
paul@paulubuntu:~/prog/projetS3/Solver$ ls
grid_00  grid_00.result  main.c  main.o  Makefile  solver  solver.c  solver.h  s
olver.o
paul@paulubuntu:~/prog/projetS3/Solver$ 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
paul@paulubuntu:~/prog/projetS3/Solver$
```

Figure 5 – Fichier « network.weights »

J'ai par la suite pris ces résultats, puis découpé des images de chiffres allant de 1 à 9 et pris une grille vide pour pouvoir les afficher dans le logiciel.

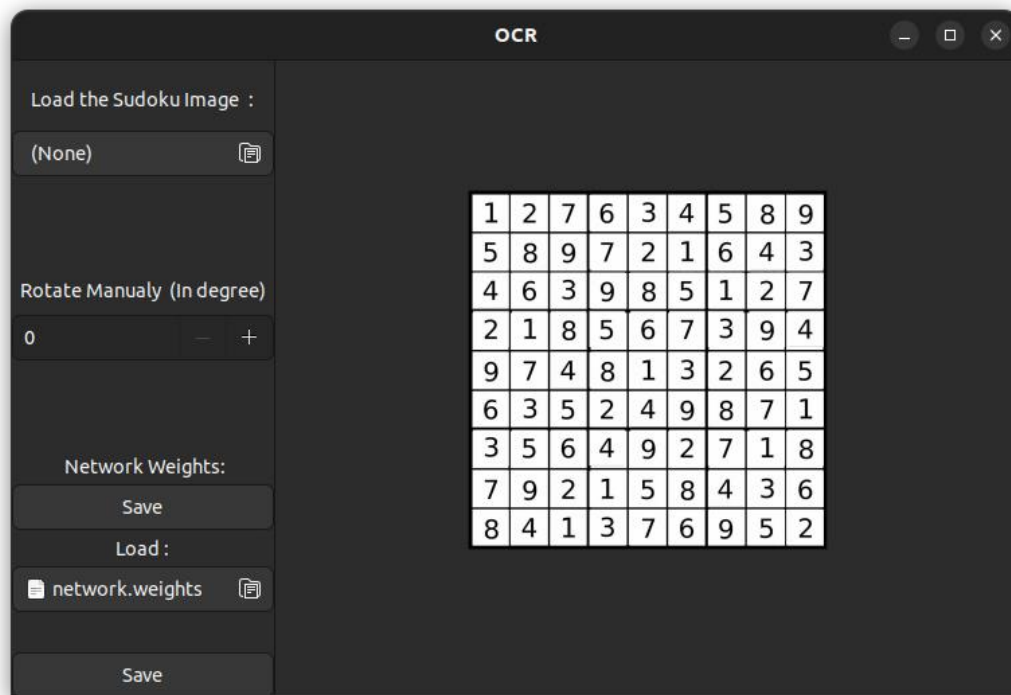


Figure 6 – Fichier « network.weights »

Voici le résultat de la grille après avoir appuyé sur le bouton « Start ». Le bouton « Start » devient alors « Save » et offre la possibilité d’obtenir une image en .png de la grille comme celle-ci :

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

Figure 7 – Image de rendus

## 2. Fabio Sintoni et Mathis Perot

Nous avons continué de travailler sur la partie détection et découpage de l'image.

Pour cela, nous avons déjà une image en noir et blanc et plus contrastée que l'originale.

Pour mieux afficher les lignes nous avons décidé de partir sur la piste des algorithmes de détection de contours utilisant les filtres de Sobel, Canny ou Prewitt.



Figure x – Exemple du filtre de Sobel

Cependant, nous pouvions déjà obtenir un résultat grâce aux images contrastées et les filtres précisés ci-dessus était trop compliqués à implémenter pour nous, c'est pourquoi nous avons finalement passé notre chemin.

Nous avons donc réalisé une première transformation d'image à l'aide de la librairie SDL. Nous avons codé un programme permettant d'augmenter les contrastes d'une image puis de la mettre en noir et blanc pour pouvoir mieux la traiter et reconnaître la grille.

Voici deux exemples d'images obtenus après cette transformation :



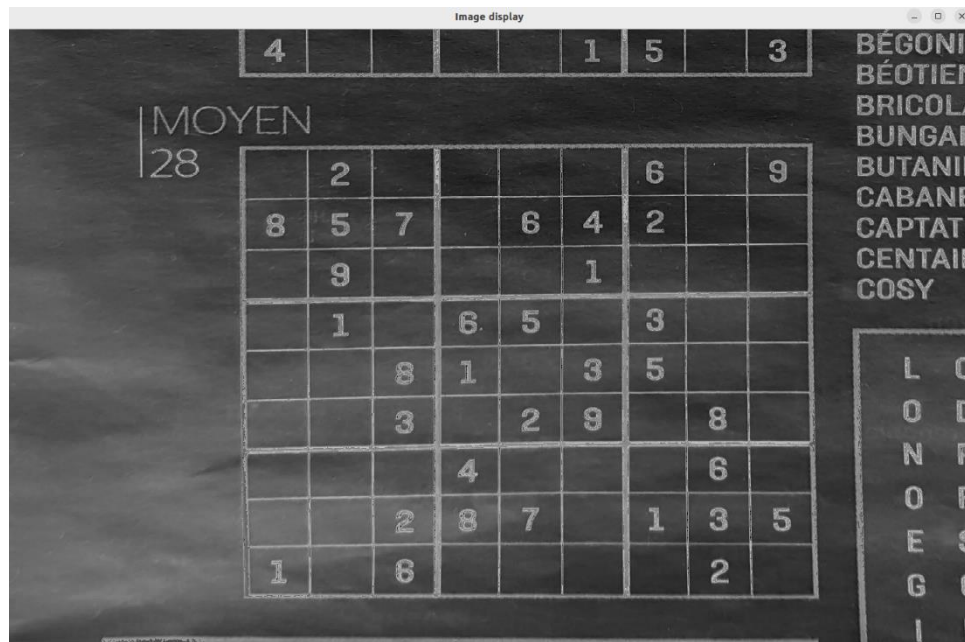


Figure x – 1<sup>er</sup> Exemple d'image après augmentation des contrastes.

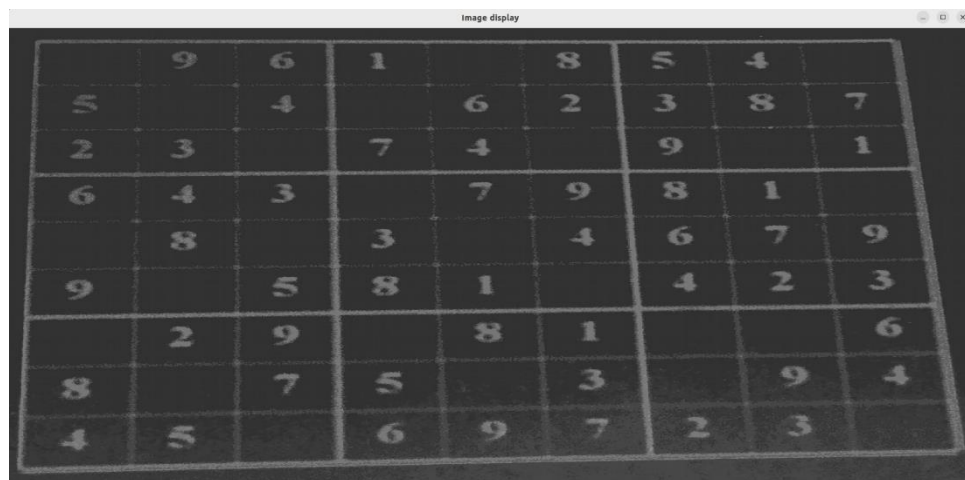


Figure x - 2<sup>ème</sup> Exemple d'image après augmentation des contrastes.

Par la suite, nous avons trouvé un algorithme du nom de “transformée de Hough” permettant de reconnaître des formes dans une image. Grâce à cela, nous pouvions enfin détecter les différentes lignes présentes et ainsi découper la grille de sudoku en différentes cases.



Nous nous sommes donc attardés sur la transformée de Hough, qui représente un moyen de stocker dans une matrice les coordonnées radiales de chaque droite. Nous avons donc commencé par créer cette matrice d'accumulation.

```
Uint32 pixel;
Uint8 r, g, b;

for (size_t x = 0; x < width; x++)
{
    for (size_t y = 0; y < height; y++)
    {
        //printf("testing pixel #%zu\n", x+y*width);
        pixel = pixels[x+y*width];
        SDL_GetRGB(pixel, surface->format, &r, &g, &b);

        //if white
        if (r+g+b >= 400)
        {
            unsigned int t = 0;
            p = x*cos(t*M_PI/180) + y*sin(t*M_PI/180);
            //printf("testing pixel #%zu to acc value %zu,%i\n",x+y*width, p, t);
            acc[p][t]++;
            t = 90;
            p = x*cos(t*M_PI/180) + y*sin(t*M_PI/180);
            //printf("testing pixel #%zu to acc value %zu,%i\n",x+y*width, p, t);
            acc[p][t]++;
        }
    }
}
```

Figure x – Capture d'écran du programme permettant de remplir la matrice de Hough en fonction des différents pixels de l'image de départ

Lorsque nous l'affichons, nous obtenons un graphe comme celui-ci:

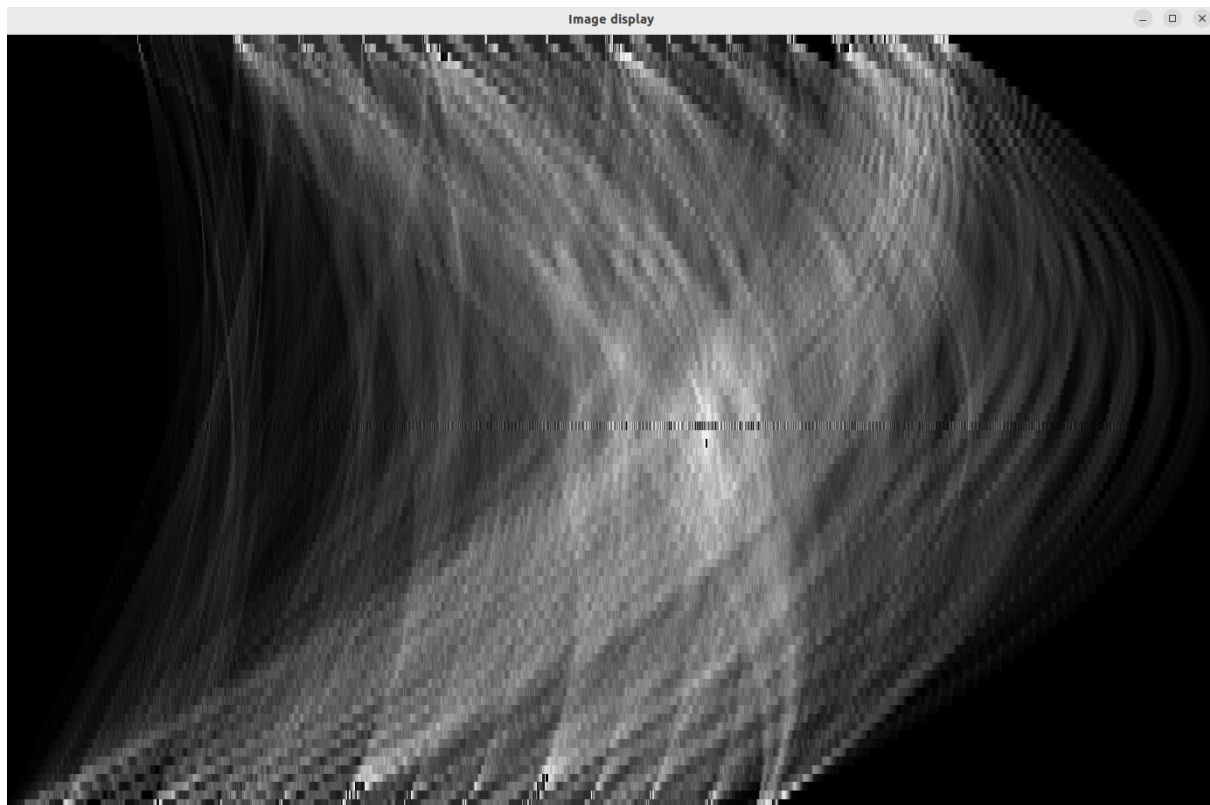


Figure x - Image obtenue représentant la matrice d'accumulation de la transformation de Hough

Les points les plus lumineux (les plus blancs) de ce graphe correspondent aux points de l'image où il y a une forte probabilité d'existence d'une ligne droite assez grande.

Plus le pixel est blanc, plus il y a de points sur la droite correspondante dans l'image de départ.

Ainsi, nous avons cherché les points les plus lumineux de cette matrice pour ensuite trouver les différents points d'intersection entre ces droites. Ces points d'intersection vont nous permettre de détecter les cases de la grille de Sudoku.

Nous réalisons donc une matrice d'intersections à l'aide de la matrice d'accumulation de Hough où tous les points blancs sont en fait les points dessinant la grille de Sudoku. Cette matrice fait en réalité la même taille que l'image de départ, nous sommes donc quasiment à la transformation finale.

```

unsigned int intersect[width*height];
unsigned int vertical[rho/9];
unsigned int horizontal[rho/9];

for(p = 0; p < rho; p += 9){
    unsigned int moyenne = 0;
    unsigned int ecart = 0;

    for(size_t i = p; i < p+9; i++){
        moyenne += acc[i][0];
    }
    moyenne /= 9;

    for(size_t i=0; i<9; i++){
        ecart += (acc[p+i][0]+moyenne)*(acc[p+i][0]+moyenne);
    }
    ecart /= 9;
    ecart = sqrt(ecart);
    if(moyenne > 600 /*&& ecart < 50*/){
        horizontal[p/9] = 1;
        //printf("j'ai mis un 1 dans horizontal");
    }

    moyenne = 0;
    ecart = 0;

    for(size_t i = p; i < p+9; i++){
        moyenne += acc[i][90];
    }
    moyenne /= 9;

```

```

        for(size_t i=0; i<9; i++){
            ecart += (acc[p+i][0]+moyenne)*(acc[p+i][0]+moyenne);
        }
        ecart /= 9;
        ecart = sqrt(ecart);
        if(moyenne > 600 /*&& ecart < 50*/){
            vertical[p/9] = 1;
            //printf("j'ai mis un 1 dans vertical");
        }
    }
}

```

Figures x et x – Captures d'écran du programme créant la matrice des points d'intersections

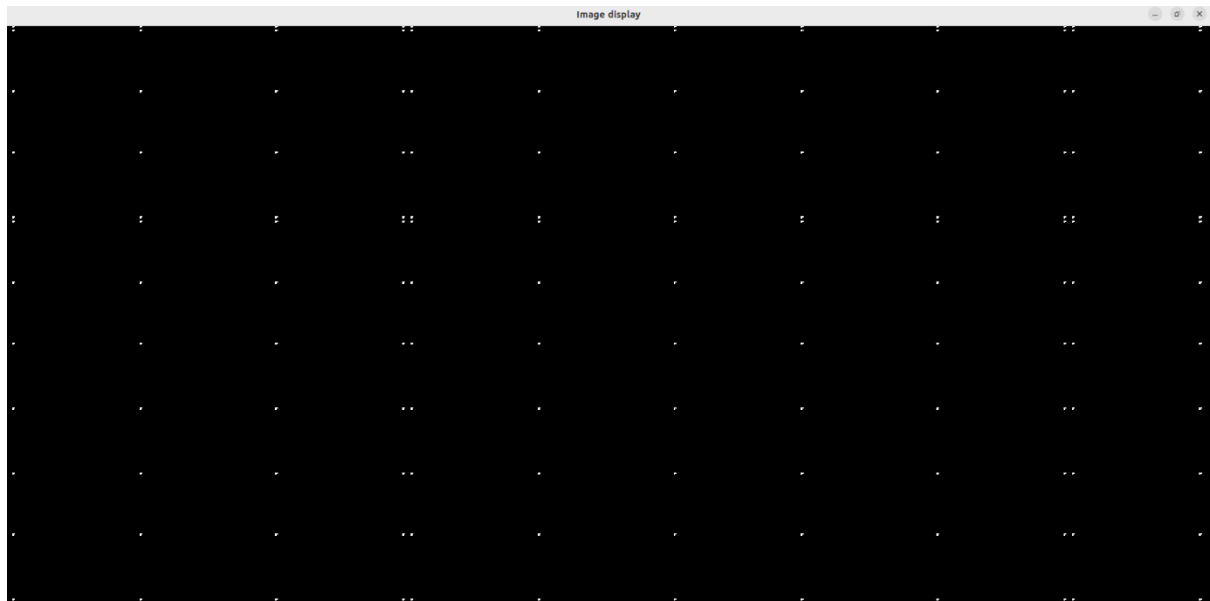
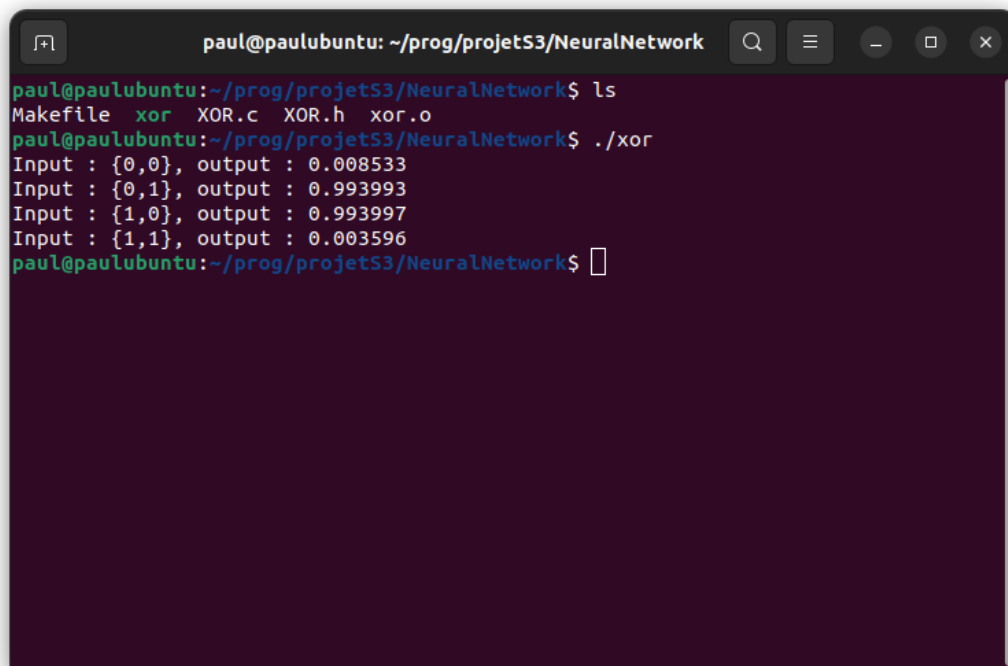


Figure x – Image représentant la matrice trouvant les points d'intersections des droites dessinant la grille.

### 3. Omri Zakaria Dakka

Nous avons dans un premier temps cherché à comprendre comment notre réseau de neurones fonctionnera. Pour cela nous avons créé un prototype de réseaux en leur apprenant l'opérateur XOR. Pour cela nous avons utilisé des fonctions d'activation de type sinus qui ont l'intérêt d'être simple à utiliser pour la propagation à l'avant et d'autres fonctions de type cosinus pour la propagation en arrière. Ce design a pu être effectué pour une fonction simple avec seulement 2 paramètres d'entrée.



```
paul@paulubuntu: ~/prog/projetS3/NeuralNetwork
paul@paulubuntu:~/prog/projetS3/NeuralNetwork$ ls
Makefile  xor  XOR.c  XOR.h  xor.o
paul@paulubuntu:~/prog/projetS3/NeuralNetwork$ ./xor
Input : {0,0}, output : 0.008533
Input : {0,1}, output : 0.993993
Input : {1,0}, output : 0.993997
Input : {1,1}, output : 0.003596
paul@paulubuntu:~/prog/projetS3/NeuralNetwork$
```

Figure x – Fonction XOR

Cependant lors de l'adaptation de ce réseau pour notre OCR nous avons rencontré deux problèmes majeurs. En effet, les fonctions sinus et cosinus sont assez longues à l'exécution et l'importante masse d'information de nos images rendait le réseau inutilisable sans attendre plusieurs heures. De plus, la fonction sinus a pour problème d'arrêter d'apprendre au bout d'un certain nombre d'itération. Nous avons donc repris notre design depuis le début.

Pour le réseau de neurones nous avons opté pour 784 neurones dans la première couche représentant la matrice de 28px par 28px qui sont la taille des images réduites en entrée. Dans la seconde couche nous avons 10 neurones ayant chacun 784 poids. Ils seront activés par la fonction relu qui retourne  $x$  si  $x > 0$  et retourne 0 sinon.

```
double relu(double x)
{
    if(x > 0)
    {
        return x;
    }
    return 0;
}
```

Figure x – Fonction relu.

Dans la dernière couche nous avons 10 neurones qui ont chacun 10 poids et qui sont activés par la fonction softmax.

```
void softmax(struct Neuron *net)
{
    double sum = 0.0;
    for(size_t i = 0; i < 9;i++)
    {
        net[i].res = exp(net[i].res);
        sum += net[i].res;
    }

    for(size_t i = 0; i < 9;i++)
    {
        net[i].res = net[i].res / sum;
    }
}
```

Figure x – Fonction softmax

Après avoir fait nos calculs nous utilisons la propagation en arrière afin de corriger les poids de notre réseau de neurones. Nous utilisons pour cela la fonction dérivée de relu qui va donc retourner 1 si  $x > 0$  et 0 sinon.

```
double relu_deriv(double x)
{
    if(x > 0)
    {
        return 1;
    }
    return 0;
}
```

Figure x – Fonction dérivée de relu

Nous corrigeons donc nos poids après et appliquons un facteur d'apprentissage de 0.01 sur le taux d'erreur que nous avons trouvé avec la propagation en arrière. À la fin nous faisons tourner notre réseau 20000 fois en à peu près 30 secondes et nous obtenions des résultats fiables à 99%. Pour tester le réseau de neurones il suffit d'appuyer sur le bouton « Start » du logiciel.

```
paul@paulubuntu:~/prog/projetS3/Software$ ./main
Network :
Resultat : In: 1, Out : 1
Resultat : In: 2, Out : 2
Resultat : In: 3, Out : 3
Resultat : In: 4, Out : 4
Resultat : In: 5, Out : 5
Resultat : In: 6, Out : 6
Resultat : In: 7, Out : 7
Resultat : In: 8, Out : 8
Resultat : In: 9, Out : 9
```

Figure x – Résultat du réseau de neurone



## **VI. Conclusion**

### 1. Paul FERREIRA

Ce projet m'aura permis d'apprendre énormément de choses. En effet, j'ai pu apprendre à créer des logiciels en C, créer des algorithmes qui résolvent des sudokus ainsi que la création automatisée d'images. J'ai pu aussi apprendre et comprendre le fonctionnement d'un réseau de neurones. J'ai aussi par la même occasion appris énormément sur le fonctionnement du langage C aussi capricieux soit-il. Ma fonction de chef de groupe m'a aussi permis d'améliorer mes compétence organisationnelles et relationnelles entre les différents membres de mon groupe.

## 2. Fabio SINTONI

Durant ce projet, j'ai pu approfondir mes notions de programmation en C avec notamment l'apprentissage de la librairie SDL. J'ai donc progressé en ce qui concerne le traitement d'images mais aussi dans la compréhension d'algorithmes complexes. En effet, la transformation de Hough n'était pas simple à comprendre au premier abord mais nous avons réussi petit à petit à nous familiariser avec pour la transcrire en langage de programmation. De plus, j'ai dû m'adapter aux différentes contraintes imposées par le cahier des charges qui nous était fourni ce qui n'a pas toujours été simple. Mais nous avons fait face aux difficultés tous ensemble pour mener à bien la réalisation de ce projet.

#### 4. Mathis Perot

J'ai pu au fil de ce projet mettre à rude épreuve ma compréhension de notions mathématiques et ma capacité à transposer ces notions dans un langage de programmation, J'ai appris à toucher du doigt les librairies Maths et SDL du langage C, et j'ai aussi une vision plus globale de comment est-ce que l'on traite une image en informatique. Je sais grâce à ce projet que le traitement de l'image est une tâche ardue qui nécessite beaucoup de sous-traitements qui sont essentiels les uns aux autres, (par exemple, pour que la transformée de Hough fonctionne correctement, il faut d'abord que les pixels visés par cette transformée soit déjà mis différents des autres, avec un filtre de Sobel, ou comme nous l'avions fait). Cependant, la difficulté que j'ai eu à comprendre et visualiser les choses que je devais faire me font dire que le domaine du traitement d'image n'est pas fait pour moi, et que je ne souhaiterais pas en faire mon métier.

Enfin, à cause du temps passé sur cette partie du projet, je n'ai pu voir comment se faisait les autres parties du projet que très partiellement, et je n'ai donc pas pu expérimenter plusieurs domaines de l'informatique comme je l'aurais souhaité.

#### 4. Omri Zakaria DAKKA

La réalisation de ce projet et plus particulièrement du réseau de neurones m'a apporté beaucoup de connaissances informatiques et m'a poussé à utiliser et à revoir des notions que je ne maîtrisais pas convenablement. La difficulté rencontrée étant bien supérieure à celle travaillée en TP, Il fallait effectuer un travail de recherche pour assimiler chacune des notions avec lesquelles je n'étais pas familier. Finalement, comme le dit le dicton "c'est en forgeant que l'on devient forgeron" et c'est en pratiquant que l'on apprend le mieux, et cela notamment dans le domaine de l'informatique.

## 5. Conclusion générale

Du fait du retard accumulé depuis la première soutenance et de la complexité de la tâche que relevait le traitement de l'image nous n'avons malheureusement pas pu finir ce projet en temps voulu. Cependant nous sommes fiers d'avoir pu finir et d'avoir rendu fonctionnel toutes les autres parties du projet que représentent le logiciel, la rotation de l'image, le réseau de neurones ou même le solveur de Sudoku. Nous estimons avoir beaucoup appris au travers la réalisation de ce projet que ce soit en termes de programmation, de compréhension du monde informatique ou même de comment travailler en équipe. Nous avons pris beaucoup de plaisir à travailler sur ce projet et nous espérons avoir réussi à transmettre ce plaisir au travers de ce document.