

OCR Rapport 2

Fait par les membres de STEL :

Ewan Schwaller, Lorenzo Taalba
Sacha Reggiani et Tristan Hette

Decembre 2022

Contents

1	Introduction	3
1.1	Présentation du groupe	3
1.1.1	Ewan SCHWALLER	3
1.1.2	Lorenzo TAALBA	3
1.1.3	Sacha REGGIANI	3
1.1.4	Tristan HETTE	4
1.2	Répartition du projet	4
1.3	Avancement du projet	5
2	Traitement de l'image	6
2.1	Couleur	6
2.2	Normalisation	7
2.3	Gamma & Contraste	9
2.3.1	Gamma	9
2.3.2	Contrast	9
2.4	Dilatation & Erosion	11
2.4.1	Dilatation	11
2.4.2	Erosion	12
2.5	Otsu	13
2.6	Rotation de l'image	15
2.6.1	Manuelle	15
2.6.2	Automatique	16
3	Découpage de l'image	17
3.1	Détection des contours	17
3.1.1	Flou Gaussian	17
3.1.2	Filtre de Sobel	17
3.1.3	Non-maximal Suppression, Double Threshold, Hysteresis	18
3.2	Détection de ligne	21
3.2.1	Algorithme de Hough	21
3.2.2	Détection des lignes	21
3.2.3	Suppressions des lignes inutiles	23
3.3	Détection des carrées	23
3.4	Détection du plus gros carré	24
3.5	Re-cadrage de l'image	25
3.6	Découpage	27
3.7	Transformation des images en données exploitables	28

4 Sudoku Solver	30
4.1 Solver brut force	30
4.2 Solver pré-traitement	30
4.3 Solver optimisé	32
5 Réseau de neurone	33
5.1 Architecture du réseau	33
5.2 Fonctionnement du réseau de neurone	33
5.3 Preuve XOR	34
5.4 Reconnaissance de caractère	35
5.4.1 Architecture global	35
5.4.2 Entrainement	36
5.4.3 Banque d'images	36
6 Interface graphique	38
6.1 Introduction	38
6.2 Structure	38
6.2.1 Nouveaux Types	38
6.2.2 Fonctions	39
6.3 Fonctions	40
6.3.1 main()	40
6.3.2 on_draw()	40
6.3.3 on_start()	41
6.3.4 on_choose_file()	41
6.3.5 on_configure()	41
6.4 Lancement	43
6.4.1 Glade	43
6.4.2 Ajout	43
6.4.3 Résultat	44
7 Conclusion	45

1 Introduction

1.1 Présentation du groupe

1.1.1 Ewan SCHWALLER

Je suis intéressé par l'informatique depuis maintenant quelque années. C'est donc pour ça que j'ai décidé de rejoindre l'EPITA juste après l'obtention de mon bac. En plus de cet intérêt, j'ai pu également commencer à programmer et à coder durant mon temps libre au lycée. J'ai donc construit quelque base en C# mais aussi dans le codage de site (avec les basiques HTML, CSS et JavaScript). Ainsi je commence le projet en ayant jamais fait de C avant cette année, mais je compte bien découvrir tous ses aspects lors de ce projet !

1.1.2 Lorenzo TAALBA

Anciennement joueur de jeux vidéo à mes temps perdus, c'est dans cette discipline que j'ai découvert l'envie de coder, changer les règles de ces univers pour créer le mien. Ce passe-temps est devenu ma passion et s'est étendu à l'informatique en général. J'ai approfondi mes connaissances avec la spécialité informatique au lycée, qui m'a permis de mieux comprendre le fonctionnement de la programmation. Cependant, je n'ai jusqu'à présent touché qu'à des langages de haut niveau et donc le C est pour moi une découverte et un nouveau défi.

1.1.3 Sacha REGGIANI

Amateur de jeux vidéo depuis tout jeune, l'informatique m'intéresse depuis maintenant la 3ème, résultant d'une curiosité d'en apprendre plus sur la création de ces œuvres. J'ai suivi des tutoriels sur internet, j'ai appris en bidouillant chez moi et en faisant des recherches. J'ai ainsi appris les bases de quelques langages comme le Python et le HTML. Jusqu'ici, la programmation était un loisir fait principalement de découvertes et de pratiques dans lesquelles je me suis bien amusé. J'ai aussi fait des recherches sur l'art du game design, toujours dans la curiosité d'en apprendre sur le jeu vidéo et sa création.

1.1.4 Tristan HETTE

J'ai commencé à m'intéresser à l'informatique en 1ère, car je suis curieux et que je trouvais que l'informatique était vraiment un domaine passionnant et plus précisément le code. J'avais l'impression que rien n'était impossible avec. Je me suis donc formé à quelques langages de programmations tels que le Python ou le C grâce à des ressources en ligne tels que Open-Classroom et SoloLearn. Je me suis également intéressé à d'autres aspects de l'informatique comme le développement de jeux vidéo car je suis un joueur depuis mon enfance et la Cyber Sécurité. Ce projet me permet de développer mes connaissances en C et également d'en apprendre plus sur les réseaux de neurones.

1.2 Répartition du projet

	Ewan	Lorenzo	Sacha	Tristan
Chargement de l'image				
Suppression des couleurs				
Pré-traitement de l'image				
Détection de la grille				
Découpage de l'image				
Algorithme de résolution d'un sudoku				
Réseau de neurones				
Interface				

1.3 Avancement du projet

Chargement de l'image	100%	Résolution du sudoku	100%
Suppression des couleurs	100%	Binarization des couleurs	100%
Traitement de l'image	100%	Interface	100%
Rotation manuelle de l'image	100%	Rotation automatique de l'image	100%
Détection de la grille	100%	Découpage de l'image	100%
Réseau de neurones (XOR)	100%	Réseau de neurones complet	100%
Reconnaissance des caractères	100%	Reconstruction de la grille	100%

2 Traitement de l'image

2.1 Couleur

Tout d'abord, nous avons passé l'image en noir et blanc. Pour se faire, on s'est servi du TP fait en programmation et on a appliqué la même méthode : on calcule la moyenne de chacune des valeurs RGB et on remplace sa couleur par cette valeur moyenne trouvée.

calcule: $\text{grey} = 0.3*\text{red} + 0.59*\text{green} + 0.11*\text{blue}$

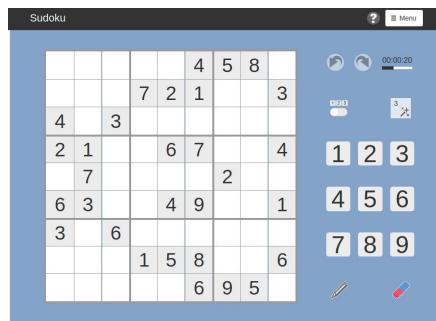


Figure 1: Avant

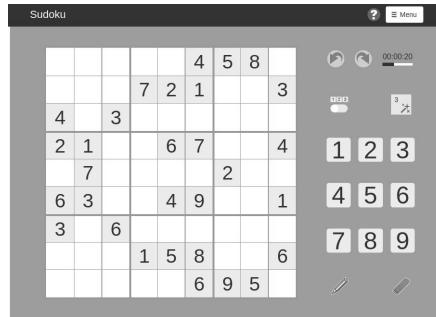


Figure 2: Après

2.2 Normalisation

Tout d'abord, nous effectuons une normalisation de l'image si besoin.

Dans le traitement des images, la normalisation est un processus qui modifie la plage des valeurs d'intensité des pixels. Les applications comprennent les photographies dont le contraste est faible en raison de l'éblouissement, par exemple. La normalisation est parfois appelée étirement du contraste ou étirement de l'histogramme. Dans des domaines plus généraux du traitement des données, comme le traitement des signaux numériques, on parle d'expansion de la plage dynamique.

L'objectif de l'expansion de la plage dynamique dans les diverses applications est généralement d'amener l'image, ou un autre type de signal, dans une plage plus familière ou normale pour les sens, d'où le terme de normalisation. Souvent, la motivation est d'obtenir une cohérence dans la gamme dynamique pour un ensemble de données, de signaux ou d'images afin d'éviter toute distraction ou fatigue mentale. Par exemple, un journal s'efforcera de faire en sorte que toutes les images d'un numéro partagent une gamme de niveaux de gris similaire.

Pour effectuer cette normalisation, nous avons d'abord trouver les pixels le plus lumineux (donc le plus blanc) et le plus sombre (donc le plus noir)

Une fois cela fait, on applique a chaque pixel la formule suivante :

$$pixels[i] = (pixels[i] - min - 30) / (max - min) * 255$$

où

$pixels[i]$: la valeur du ième pixel

max : la valeur du pixel le plus lumineux

min : la valeur du pixel le plus sombre

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

Figure 3: Avant

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

Figure 4: Après avoir effectué la normalisation

2.3 Gamma & Contraste

2.3.1 Gamma

Nous allons ensuite faire une légère correction du gamma de l'image.

Pour se faire, nous appliquons a chaque pixel :

$$pixels[i] = (pixels[i]/255)^{0.9} * 255$$

où pixels[i] : la valeur du ieme pixel

2.3.2 Contrast

On va faire de même pour le contraste et appliquer cette formule a chaque pixel :

$$pixels[i] = 3 * (pixels[i] - 128) + 128$$

où pixels[i] : la valeur du ieme pixel

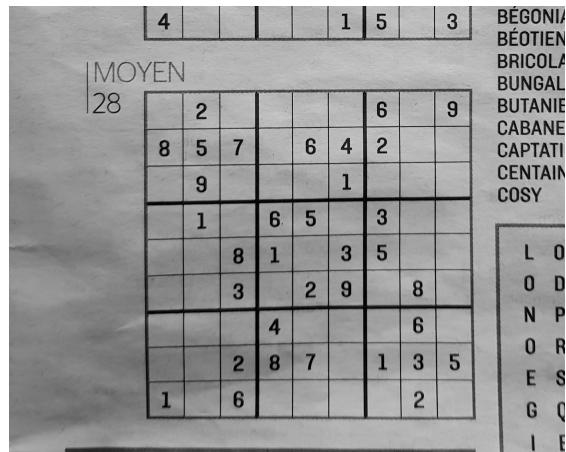


Figure 5: Avant

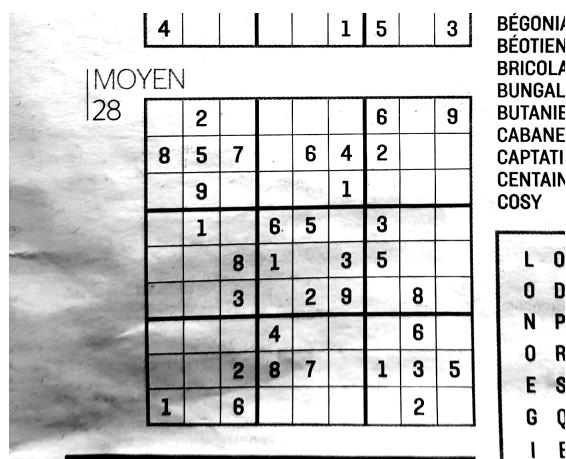


Figure 6: Après avoir appliqué la correction du gamma et du contraste

2.4 Dilatation & Erosion

L'objectif de cette partie est d'enlever le bruit de l'image pour faire en sorte que la reconnaissance de la grille et des chiffres plus facile.

2.4.1 Dilatation

On commence par la dilatation de l'image, qui agrandit les détails de l'image et donc rends les lignes et les chiffres plus claire.

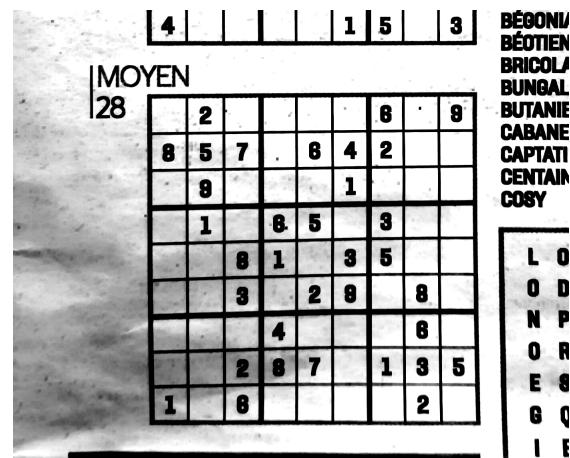


Figure 7: Dilatation

2.4.2 Erosion

Ensuite, on l'érode, pour enlever la largeur ajouter aux lignes et aux chiffres pour retrouver leurs tailles d'origines.

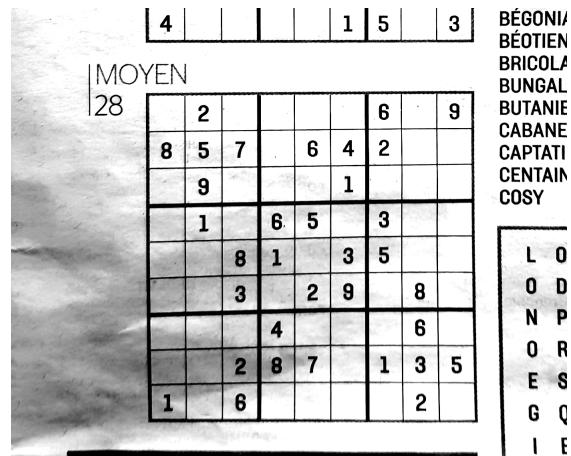


Figure 8: Erosion

2.5 Otsu

La méthode d’Otsu consiste en 3 étapes:

- Obtenir l’histogramme de l’image (distribution des pixels)
- Calculer la valeur seuil T
- Remplacer les pixels de l’image en blanc dans les régions où la saturation est supérieure à T et en noir dans les cas contraires.

Pour la première étapes, on créée une liste de longueur 256, qui va donc être notre histogramme, et on itère dans chaque pixels en prenant sa valeur (allant de 0 à 255) et on incrémenté l’élément correspondant de 1.

Par exemple, si un pixel vaut 42, on incrémenté la 42ème valeur de 1.

Ensuite, on calcule le seuil grâce à Otsu.

Enfin, pour chaque pixel on remplace en blanc si la saturation est supérieure à T et en noir dans les cas contraires.

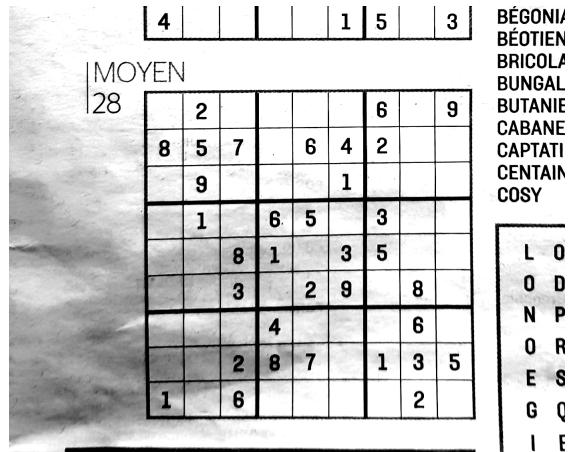


Figure 9: Avant

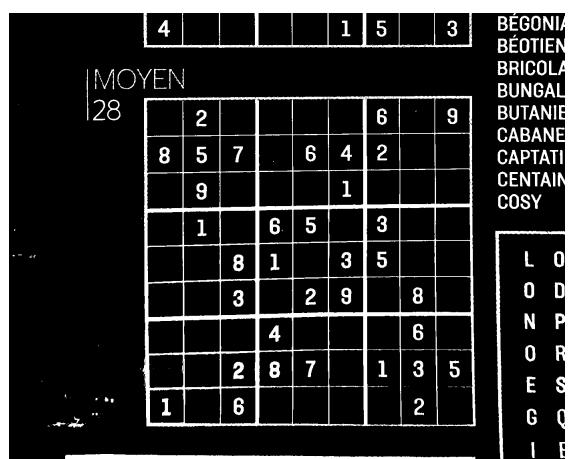


Figure 10: Après avoir appliqué Otsu

2.6 Rotation de l'image

2.6.1 Manuelle

Pour effectuer une rotation à l'image avec un angle donné, on applique une matrice de rotation à l'image. En effet, cette méthode donne des rapides et bons résultats.

La matrice s'exprime avec les formules suivantes pour trouver les coordonnées (x', y') d'un pixel tourné, donc se trouvant sur l'image final, tourné sur un angle θ ayant comme origine le pixel situé en (x, y)

$$x' = x * \cos(\theta) - y * \sin(\theta)$$
$$y' = x * \sin(\theta) + y * \cos(\theta)$$

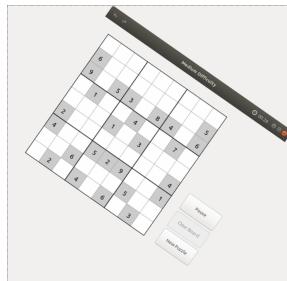


Figure 11: Avant

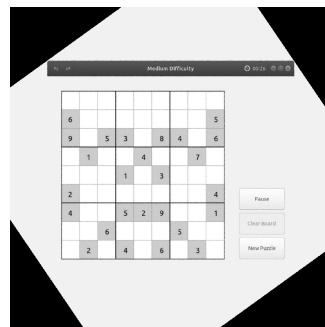


Figure 12: Après avoir effectué une rotation de 35° dans le sens anti-horaire

2.6.2 Automatique

Pour trouver l'angle de rotation à faire automatiquement, on s'est servi de l'algorithme détectant les ligne. En effet, une fois l'angle trouvé, nous avions plus qu'à appliquer la fonction dont on s'est servi pour la rotation manuelle.

Pour trouver l'angle de rotation, nous avons pris l'angle θ le plus présent dans l'accumulateur de Hough puis appliquons sa valeur à l'image.

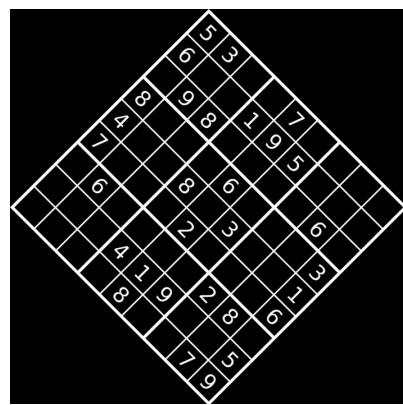


Figure 13: Avant

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

Figure 14: Après avoir effectué une rotation automatique

3 Découpage de l'image

Pour cette partie on a procédé ainsi :

- Détection des contours
- Détection des lignes
- Détection des carrées
- Détection du plus gros carré (donc la grille)
- Re-cadrage de l'image pour n'avoir plus que la grille
- Découpage de la grille

3.1 Détection des contours

3.1.1 Flou Gaussian

Pour détecter les contours, il faut tout d'abord appliquer un flou gaussian a l'image.

Ce filtre agit sur chaque pixel de l'image en réglant sa valeur sur la moyenne de tous les pixels présents dans un rayon défini. Plus ce rayon est élevé, plus le flou sera important.

3.1.2 Filtre de Sobel

L'opérateur utilise des matrices de convolution. La matrice de taille 3×3 subit une convolution avec l'image pour calculer des approximations des dérivées horizontale et verticale.

Soit A l'image source, G_x et G_y deux images qui en chaque point contiennent des approximations respectivement de la dérivée horizontale et verticale de chaque point. Ces images sont calculées comme suit:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * A \text{ et } G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

En chaque point, les approximations des gradients horizontaux et verticaux peuvent être combinées comme suit pour obtenir une approximation de la norme du gradient :

$$G = \sqrt{G_x^2 + G_y^2}$$

On peut également calculer la direction du gradient comme suit :

$$\theta = \text{atan2}(G_x, G_y)$$

3.1.3 Non-maximal Suppression, Double Threshold, Hysteresis

Ces trois algorithmes, appliqués à la suite les uns à la suite des autres, vont permettre de créer une image où chaque pixel blanc représente un contour possible.

Non-maximal Suppression

La suppression non maximale est une méthode de vision par ordinateur qui sélectionne une seule entité parmi de nombreuses entités qui se chevauchent (par exemple les boîtes englobantes dans la détection d'objets). Le critère est généralement d'écartier les entités qui sont en dessous d'une limite de probabilité donnée.

Idéalement, l'image finale devrait avoir des bords fins. Il faut donc effectuer une suppression non maximale pour amincir les bords.

Le principe est simple : l'algorithme passe en revue tous les points de la matrice d'intensité du gradient et trouve les pixels ayant la valeur maximale dans les directions des bords.

Double Threshold

Le double seuil vise à identifier 3 types de pixels : forts, faibles et non pertinents :

- Les pixels forts sont des pixels qui ont une intensité si élevée que nous sommes sûrs qu'ils contribuent à l'arête finale.

- Les pixels faibles sont des pixels dont la valeur d'intensité n'est pas suffisante pour être considérée comme forte, mais pas assez faible pour être considérée comme non pertinente pour la détection des bords.
- Les autres pixels sont considérés comme non pertinents pour le bord.

Hystérésis

Sur la base des résultats du seuil, l'hystérésis consiste à transformer les pixels faibles en pixels forts, si et seulement si au moins un des pixels autour de celui qui est traité est un pixel fort.

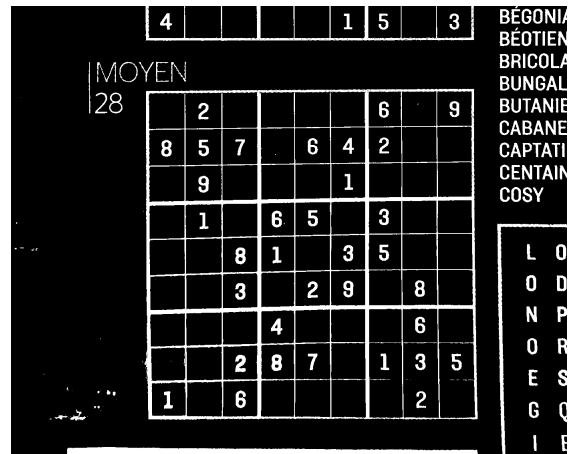


Figure 15: Avant

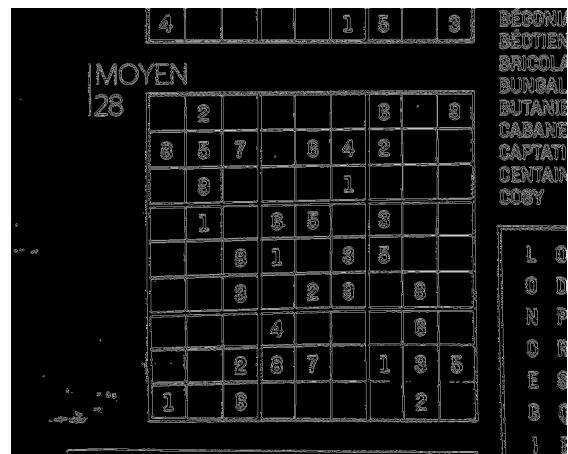


Figure 16: Après avoir appliqué Canny

3.2 Détection de ligne

3.2.1 Algorithme de Hough

Grâce l'algorithme de Hough, ou la transformée de Hough, on a pu faire en sorte que notre projet détecte les lignes. Cet algorithme construit une courbe pour chaque pixels blancs dans son accumulateur (les edge points). Cet courbe s'écrit de la forme :

$$\rho = x * \cos(\theta) + y * \sin(\theta)$$

avec θ allant de -90 deg a 90 deg.

Ainsi l'endroit où deux courbes se croisent dans l'accumulateur, représente potentiellement une ligne sur l'image.

3.2.2 Détection des lignes

Ensuite, on divise l'accumulateur en plusieurs parties et on retient, dans chacune de ces parties, la valeur maximale dépassant un seuil fixé à 30% de la valeur maximale dans tout l'accumulateur. Ces valeurs une fois obtenus, nous donnent les lignes de notre image.

Pour les trouver, on a besoin de deux points. Pour le premier, c'est facile, il suffit de transformer l'équation $\rho = x * \cos(\theta) + y * \sin(\theta)$ en cartésien. On a donc pour un couple (x_1, y_1)

$$\begin{aligned}x_1 &= \rho * \cos(\theta) \\y_1 &= \rho * \sin(\theta)\end{aligned}$$

Pour l'autre point il suffit projeter le point à une distance donnée d

$$\begin{aligned}x_2 &= \rho * \cos(\theta) + d * (-\sin(\theta)) \\y_2 &= \rho * \sin(\theta) + d * \cos(\theta)\end{aligned}$$

Ainsi, on obtient une liste dont chacune des lignes contiennent 4 valeurs décrivant deux points.

	4				1	5	3	BÉGONI
								BÉOTIEN
								BRICOLA
								BUNGAL
								BUTANIE
								CABANE
								CAPTATI
								CENTAIN
								COSY
MOYEN	28	2			6	4	2	L O
		8	5	7	6	4	2	O D
		9			1			N P
		1		6	5	3		O R
			8	1	3	5		E S
			3		2	9	8	G Q
				4			6	I E
				2	8	7	1	
				1	6		3	
							5	
							2	

Figure 17: Avant

	4				1	5	3	BÉGONI
								BÉOTIEN
								BRICOLA
								BUNGAL
								BUTANIE
								CABANE
								CAPTATI
								CENTAIN
								COSY
MOYEN	28	2			6	4	2	L O
		8	5	7	6	4	2	O D
		9			1			N P
		1		6	5	3		O R
			8	1	3	5		E S
			3		2	9	8	G Q
				4			6	I E
				2	8	7	1	
				1	6		3	
							5	
							2	

Figure 18: Après

3.2.3 Suppressions des lignes inutiles

Ensuite, on supprime les lignes inutiles afin d'optimiser la suite du programmes.

	4			1	5	3	BÉGONIA BÉOTIEN BRICOLA BUNGAL BUTANIE CABANE CAPTATI CENTAIN COSY
MOYEN							
28	2			6		9	
	8	5	7	6	4	2	
	9			1			
	1		6	5	3		L O
		8	1	3	5		O D
		3		2	9	8	N P
			4			6	O R
		2	8	7	1	3	E S
	1	6				2	G Q
							I E

Figure 19: Après

3.3 Détection des carrées

Pour la détection des carrées, on a tout d'abord calculé les équations de chaque ligne notées sous la forme :

$$\begin{cases} y = ax + b & \text{si la ligne n'est pas vertical} \\ x = a & \text{sinon} \end{cases}$$

Chacune des lignes sont stockées dans une liste pour pouvoir y accéder plus tard.

Une fois les équations obtenues, on doit trouver quatre lignes perpendiculaires entre elles. Pour savoir si deux lignes sont perpendiculaires, il y a trois possibilités :

- **Si aucune des deux lignes sont verticales** : on multiplie leur coefficients directeurs et regarde si le résultat est égale à -1.
- **Si une des deux lignes sont verticales** : on vérifie que le coefficients de la ligne non verticale est égale a 0.
- **Si les deux lignes sont verticales** : elles ne sont pas perpendiculaires

Une fois avoir trouvé quatre lignes perpendiculaires, on a bien trouvé un carré !

3.4 Détection du plus gros carré

Pour savoir quel est le plus grand, on a décidé de calculer son périmètre. Si ce dernier est plus grand que le précédent plus grand, alors, on stocke la valeur de son périmètre et les index des lignes le composant selon l'ordre dans lequel les lignes sont stockées dans la liste.

Pour le calcul du périmètre, on calcule chaque intersection entre les lignes (donc chaque coins du carrés) puis on additionne chacune de distances entre les intersections.

	4			1	5	3		BÉGONIA BÉTIEN BRICOLA BUNGAL BUTANIE CABANE CAPTATI CENTAIN COSY
MOYEN								
28	2			6	9			
	8	5	7	6	4	2		
		9		1				
	1	6	5	3				
		8	1	3	5			L O
		3	2	9	8			O D
			4		6			N P
			2	8	7	1	3	O R
			1	6		2		E S
								G Q
								I E

Figure 20:

3.5 Re-cadrage de l'image

Le re-cadrage ne consiste "qu'à" trouver quel points se situe où (En haut à droite, en bas à gauche, etc...). Pour se faire, on regarde selon leur valeur en abscisse et en ordonnée. Par exemple, celui qui aura la plus petite abscisse et la plus petite ordonnée sera en haut à gauche, etc...

Une fois l'ordre trouvé, il suffit de calculer la nouvelle longueur de l'image (c'est-à-dire la longueur de la grille) et sa nouvelle hauteur (donc la hauteur de la grille). Enfin on applique une fonction SDL pour replacer la grille correctement puis on ajuste ses dimensions selon les valeurs trouvées précédemment.

	4				1	5	3	BÉGONIA BÉOTIEN BRICOLA BUNGAL BUTANIE CABANE CAPTATI CENTAIN COSY
MOYEN		2				6		
28	8	5	7		6	4	2	
	9				1			
	1		6	5		3		
		8	1		3	5		
		3		2	9		8	
			4				6	
			2	8	7		1	3
	1		6				2	5

Figure 21: Avant

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

Figure 22: Après

3.6 Découpage

Une fois que toutes les étapes précédentes ont été effectuées, il est très simple de découper la grille. Comme l'image à déjà été redimensionnée il n'y aucune détection de grille à faire, donc on découpe l'image en 81 carrés. Pour se faire on prend la longueur de l'image (largeur ou hauteur peu importe vu que c'est un carré), et on divise cette valeur par 9 pour obtenir la taille de chaque case du sudoku. Enfin on utilise une fonction de SDL pour copier la surface désirée dans le tableau qui contiendra toutes les surfaces.

3.7 Transformation des images en données exploitable

Une fois que toutes les images ont été stockées dans une liste, il est maintenant nécessaire de les transformer afin de pouvoir les envoyer sous une forme que Tristan pourra traiter dans son réseau de neurones. Pour ce faire, nous allons parcourir toutes les images présentes dans la liste, puis parcourir chaque pixel de l'image pour l'envoyer dans un fichier texte qui aura soit des 0 pour symboliser les pixels blancs, soit des 255 pour symboliser les pixels noirs, en espaçant chaque case par un retour à la ligne.

Maintenant que le fichier texte est prêt, il ne reste plus qu'à l'envoyer à Tristan afin qu'il traite la grille de sudoku qui est stockée dans le fichier texte et détecte ainsi chaque chiffre sur chaque colonne.

5



```
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  
0,0,0,0,0,0,0,0,0,0,255,255,0,0,0,  
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  
0,0,0,0,0,0,255,255,0,0,0,0,0,0,  
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  
0,0,255,255,0,0,0,0,0,0,0,255,255,  
255,255,255,255,255,255,255,0,0,0,  
0,0,0,0,0,0,255,255,0,0,0,0,0,0,  
0,255,255,255,255,255,255,255,255,  
255,0,0,0,0,0,0,0,0,0,255,255,0,  
0,0,0,0,0,0,255,255,0,0,0,0,0,0,  
0,0,0,0,0,0,0,0,0,0,255,255,0,0,0,  
0,0,0,0,0,255,255,0,0,0,0,0,0,0,0,  
0,0,0,0,0,0,255,255,0,0,0,0,0,0,0,  
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,  
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
```

Figure 23: transformation d'une case vers un fichier texte (rognée)

4 Sudoku Solver

Le but du Sudoku Solver, doit être de pouvoir lire un fichier représentant un sudoku, le convertir dans un programme informatique, le résoudre, puis enfin enregistrer le résultat dans un autre fichier.

Tout d'abord nous avons commencé par faire le sudoku solver, ayant déjà fait un l'année précédente il était facile de le refaire en C.

4.1 Solver brut force

Le but d'une résolution par force brute est de tester toutes les possibilités jusqu'à ce que cela fonctionne. Dans le cas d'un sudoku solver cela se traduit par tester pour chaque case vide un chiffre, vérifier si la grille est toujours une grille valide, si oui passer à la case suivante, sinon tester avec le chiffre suivant et ainsi de suite jusqu'à la dernière case.

Cette fonction est très simple à mettre en place et est très courte, mais représente un problème quand à la performance du programme, celui-ci devant tester toutes les possibilités chaque fois, même celles qui sont évidentes de ne pas fonctionner.

4.2 Solver pré-traitement

Pour optimiser et contrer ce problème, nous avons revu notre implémentation du sudoku et ajouté une fonction de pré-traitement. Dans notre premier solver, notre grille était représenté par une liste simple de 81 cases, les 9 lignes * les 9 colonnes, où chaque case était un Uint (soit 0 pour une case vide, ou sinon un chiffre de 1 à 9). Dans la nouvelle implémentation, la grille est une liste de liste, une liste de 81 cases contenant chacune une liste de 10 éléments, dans ces 10 éléments, le premier représente le chiffre de la case (0 pour une case vide ou un chiffre de 1 à 9), les 9 autres étant les chiffres possibles, dans le cas d'une case libre (une liste de 1 à 9 où si la valeur du chiffre n'est pas valide par rapport aux règles du sudoku, elle est remise à 0). Autrement dit, une case:

- [0,1,2,3,4,5,6,7,8,9] veut dire que la case est vide et qu'on ne possède pas assez d'informations pour en déduire des chiffres impossibles.
- [4,0,0,0,0,0,0,0,0] que la case a été initié à 4
- [0,1,2,0,4,0,0,0,8,9] que la case est vide et que seules les chiffres 1, 2, 4, 8 et 9 peuvent être des valeurs possibles

Initialement les cases peuvent être soit dans le cas d'une case non-vide avec une valeur non-nulle au premier index et la suite à 0, soit pour une case vide avec 0 au début et les chiffres de 1 à 9 dans le reste de la liste. Par la suite nous appliquons une fonction pour supprimer les valeurs impossibles des cases libres. Le programme liste toutes les valeurs déjà rempli sur la ligne (et respectivement sur les colonnes et les zones de 3X3) et retire ces éléments de chacune des cases libres car une ligne (de même pour les colonnes et les zones de 3X3) ne peut contenir plusieurs fois le même chiffre.

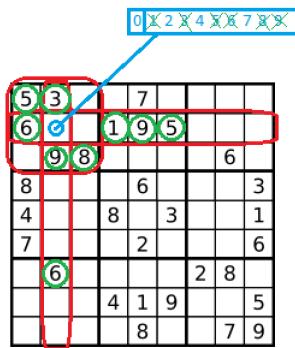


Figure 24: exemple de traitement pour une case

Grâce à cette méthode, nous évitons de tester à chaque fois des chiffres qui sont incontestablement impossibles, avec une simple boucle au tout début du programme. Cela a permis de réduire le temps de résolution de moitié par rapport au forçage brut.

```

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

Solver pré-traitement
Init finished in 812 millisec
Solve finished in 1273 millisec and code 1      1285 ms

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

Solver force brute
Init finished in 803 millisec
Solve finished in 2066 millisec and code 1      2069 ms

```

Figure 25: comparaison avec la force brute

4.3 Solver optimisé

La dernière méthode rajoute par contre, le besoin d'une capacité de stockage plus grande, alors tout d'abord nous avons directement penser à changer le fait d'avoir une liste de liste de Uint qui prenait trop de place ($81 * 10 * 4$ bytes), par une liste de liste de char qui réduirait par 4 le besoin de stockage, un char ne prenant qu'un bytes par rapport aux 4 bytes du Uint.

Ensuite lors du pré-traitement, beaucoup de cases se retrouvent avec seulement qu'un chiffre possible, mais pour le programme la case est vide et doit à chaque récursion rechercher ce chiffre dans la suite de la liste. Pour améliorer ceci, lors du pré-traitement, si une case libre ne possède plus qu'une seule valeur possible, alors il transforme cette case libre en case remplie, permettant au programme de résolution de directement passer à la case suivante. Avec ces petites modifications, le solver a réduit son temps de résolution par 10 par rapport au solver par force brute.

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

Solver optimisé
Init finished in 020 millisec
Solve finished in 156 millisec and code 1      176 ms

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

Solver force brute
Init finished in 003 millisec
Solve finished in 1537 millisec and code 1     1540 ms
```

Figure 26: comparaison avec la force brute

5 Réseau de neurone

5.1 Architecture du réseau

Notre réseau de neurone a une architecture assez simple et classique. Nous disposons de plusieurs couches, la première est la couche contenant les entrées, la dernière contient le neurone de sortie et entre ces deux couches, nous avons les couches cachées qui contiennent nos neurones.

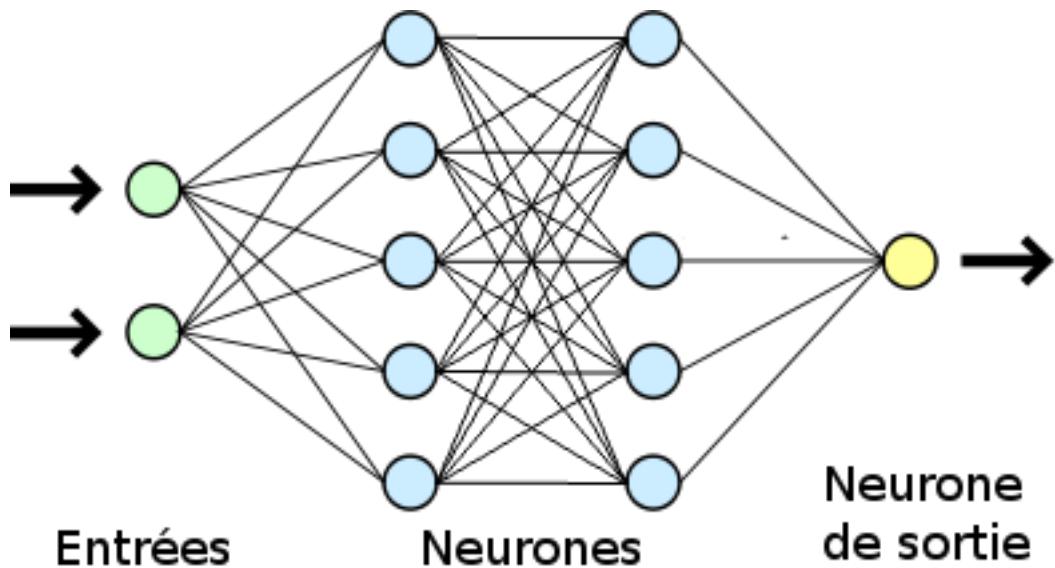


Figure 27: Architecture de notre réseau

5.2 Fonctionnement du réseau de neurone

Le fonctionnement de notre réseau de neurone est réparti en différentes parties. Tout d'abord, nous initialisons les coefficients (poids) de notre réseau avec des valeurs aléatoires. Ensuite nous avons une partie front-propagation qui permet de prédire à notre réseau une valeur de sortie grâce à la fonction d'activation sigmoïde (voir ci-dessous).

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Figure 28: Fonction sigmoïde

Et enfin nous avons la back-propagation qui elle permet à notre réseau de corriger ses erreurs grâce à la dérivée de la fonction Sigmoïde (voir ci-dessous).

$$\sigma'(x) = \sigma(x)(1 - \sigma(x))$$

Figure 29: Dérivée de la fonction sigmoïde

5.3 Preuve XOR

Pour le problème XOR, nous utilisons dans notre réseau de neurone 2 entrées, 1 couche cachées comprenant 2 neurones et 1 sortie (voir ci-dessous). Les nombres utilisés sont choisis par rapport à la table XOR (voir ci-dessous). Après de nombreuses itération, la sortie attendu doit converger vers 0 si le résultat attendu est 0 et doit converger vers 1 si le résultat attendu est 1.

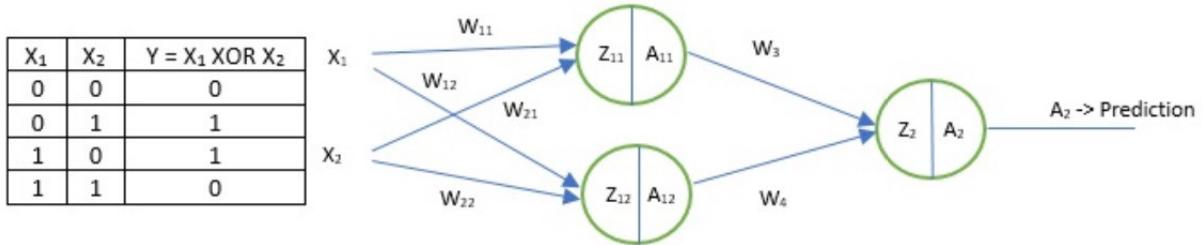


Figure 30: Table XOR et le réseau de neurone

5.4 Reconnaissance de caractère

Pour la reconnaissance de caractère nous utilisons les images $28 * 28$ renvoyer par la fonction qui coupe le sudoku en 81 sous images.

5.4.1 Architecture global

Notre réseau de neurones est composé de 3 parties comme on nous l'avons vu précédemment, dans la première partie nous avons les entrées (les images), il y a donc 784 entrées car nous avons besoin de chaque pixel de l'image, en effet, $28 * 28 = 784$ (voir ci-dessous).

Afin de rendre plus performant notre réseau de neurone, nous avons ajouter à la dernière couche la fonction softmax et nous avons également un taux d'apprentissage qui permet de contrôler la convergence du réseau afin qu'il y est un bonne équilibre entre le taux d'apprentissage et le nombre de couche cachées.

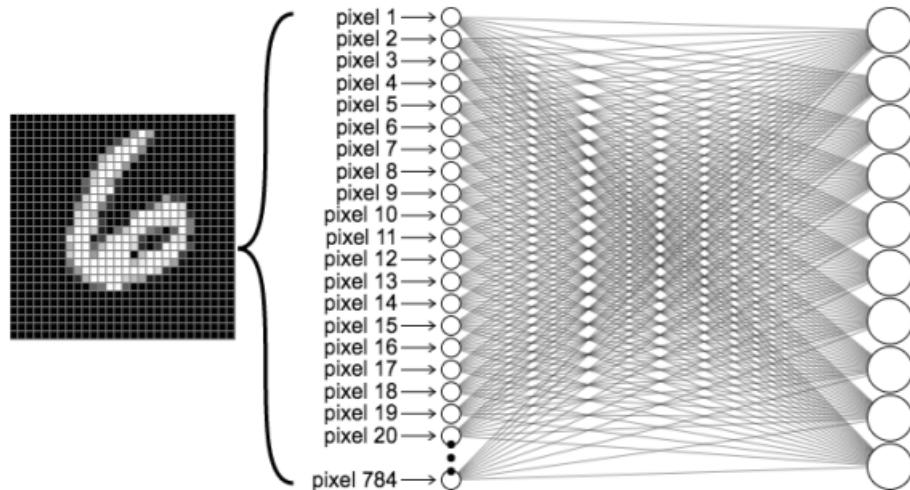


Figure 31: Entrées du réseau de neurone

5.4.2 Entrainement

Dans notre réseau neuronal, nous utilisons d'abord la propagation vers l'avant, nous calculons notre sortie en fonction du poids actuel en utilisant séquentiellement le produit scalaire et en appliquant la fonction d'activation sigmoïde à notre matrice, puis nous calculons nos erreurs, les erreurs de sortie sont obtenues en divisant la sortie finale par la sortie attendue.

Ensuite, nous utilisons la propagation arrière pour ajuster notre poids de sortie et notre poids caché en utilisant la descente de gradient, puis le réseau est finalement entraîné.

5.4.3 Banque d'images

Pour notre banque d'image, nous avons utilisé les nombreuses images du Mnist (voir ci-dessous) afin de les donner à notre réseau neuronal.

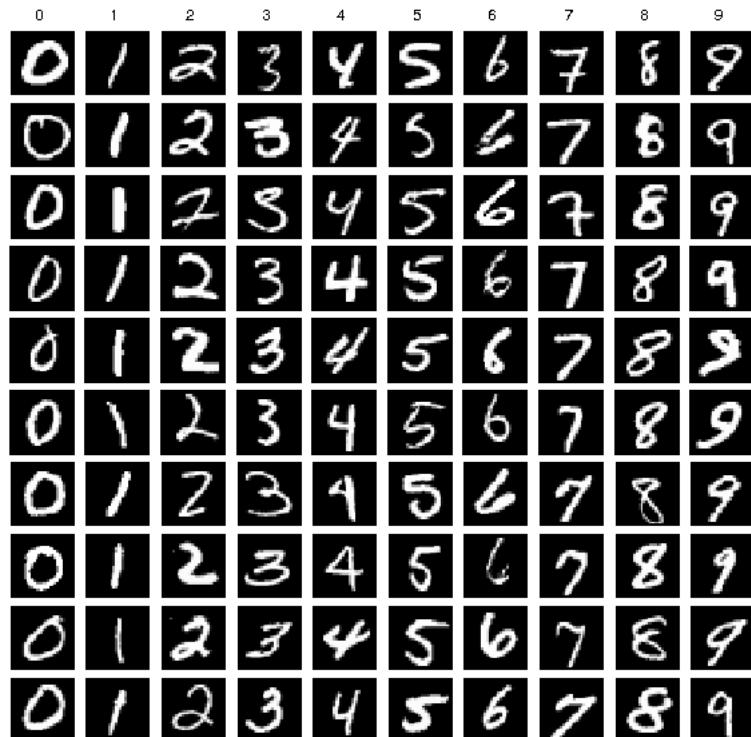


Figure 32: Banque image Mnist

6 Interface graphique

6.1 Introduction

Nous utiliserons la librairie GTK pour l'aspect graphique de l'application, celle-ci permettant d'afficher des fenêtres et de pouvoir interagir avec. Avant tout implémentation, il faut savoir ce dont on aura besoin sur l'application.

Dans notre cas il lui faut:

- Un bouton qui ouvrira un explorateur de fichier pour ouvrir les images
- Un bouton pour exécuter le solver
- Un espace pour afficher le résultat

Par la suite on y ajoutera:

- Le nom du groupe
- Un court manuel d'utilisation

6.2 Structure

6.2.1 Nouveaux Types

Dans un premier temps on implémentera plusieurs types:

- **Un type SDK:**
(qui représente le sudoku)
 - Avec un booléen qui indique si le sudoku est résolu
 - Une grille de int où sera sauvegardé la grille du sudoku avant la résolution de cette dernière
 - Une deuxième grille de int où sera enregistré la grille du sudoku résolue
 - Un deuxième booléen qui indique si le sudoku est affiché à l'écran

- **Un type UI:**

(qui représente l'interface graphique)

- Avec la Window principale
- Un DrawingArea où sera affiché le résultat
- Un Button pour lancer le programme de résolution
- Un Button qui ouvrira un explorateur de fichier
- Un int qui représente la taille de la fenêtre

- **Un type APK:**

(qui regroupe tout ce dont a besoin l'application)

- Un type UserInterface
- Un type SDK
- Une string File

6.2.2 Fonctions

Ensuite nous aurons aussi besoin de plusieurs fonctions:

- On_draw qui affiche le sudoku sur la fenêtre
- On_configure qui recadre le sudoku
- On_choose_file qui met à jour l'image sélectionnée
- On_start qui lance le programme de résolution
- main qui initialise tout

6.3 Fonctions

6.3.1 main()

Dans le main on devra initialiser aussi bien tout ce dont a besoin GTK, ainsi qu'initialiser une variable de la nouvelle structure APK qui nous sera bien utile tout au long du reste du programme. Suite à cela on doit lier les signales d'évènements aux autres fonctions et cela grâce à g_signal_connect.

6.3.2 on_draw()

C'est dans cette fonction qu'on dessinera le sudoku résolu. Pour cela on aura besoin de plusieurs choses: la taille de la fenêtre actuelle et les deux grilles de sudoku (une résolu et l'autre avant exécution du programme). Ensuite avec Cairo on peut tracer les quatre bordures de la grille en gris foncé, puis on fait une boucle pour dessiner toutes les lignes, colonnes et chiffres tout en prenant en compte de vérifier si le chiffre qu'on écrit est nouveau ou pas, dans ce cas là on changera la couleur pour l'écrire en vert.

Après cela, notre sudoku est affiché et ressemble à cela:

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 34: Affichage du sudoku

6.3.3 on_start()

C'est ici qu'on résoudra le sudoku. En premier lieu, on regarde si on a bien sélectionné une image, si c'est le cas on la chargera, l'enverra au traitement d'image, par la suite elle passera par le réseaux de neurones qui la sauvegardera dans un fichier texte. Ici elle sera résolu et mise à jour. C'est par ce biais qu'on l'a récupérera dans des fichiers textes et la sauvegardera dans nos grilles. Enfin on n'aura plus qu'à l'afficher.

6.3.4 on_choose_file()

Cette fonction ne fais rien de plus que de mettre à jour l'argument File de l'APK, pour pouvoir ensuite démarrer le programme principal.

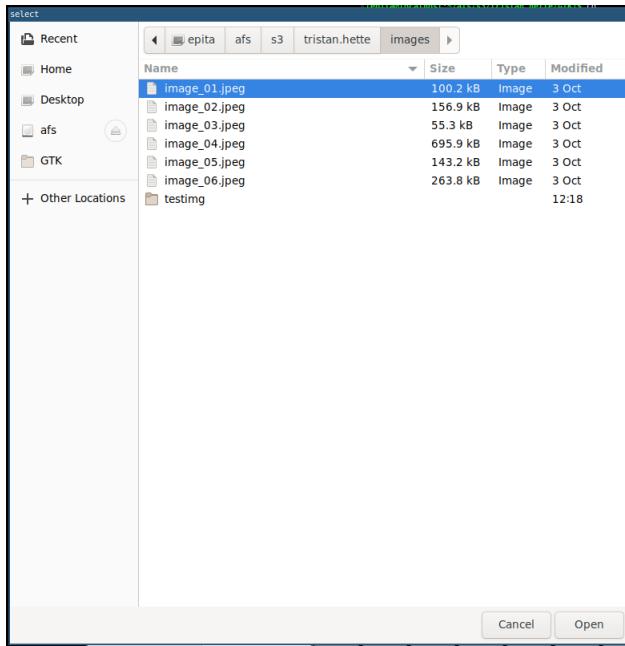


Figure 35: Explorateur de fichier

6.3.5 on_configure()

Cette fonction permet de recadrer l'affichage en mettant à jour l'argument *size* de l'UI. Cela permet à la fonction *on_draw* de ne pas dépasser la fenêtre et d'adapter la taille des cellules et des chiffres en conséquence.



Figure 36: Recadrage portrait



Figure 37: Recadrage paysage

6.4 Lancement

6.4.1 Glade

Il ne reste plus qu'à éditer l'aspect graphique de l'application avec Glade. Avec son interface simple, il a été facile d'y associer chaque élément et d'y ajouter des emplacements pour écrire le nom de notre groupe ainsi qu'un bref mode d'emploi.

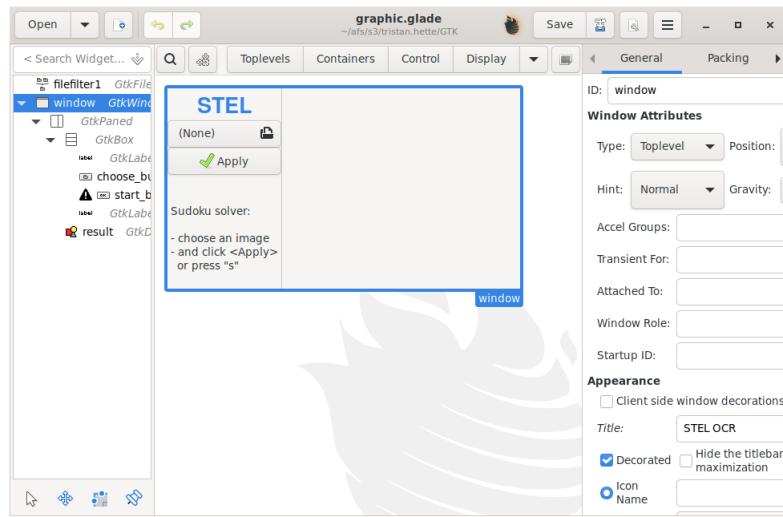


Figure 38: L'éditeur Glade

6.4.2 Ajout

On a pu de même rajouter deux petites options. La première est un filtre ajouté dès le fichier Glade permettant de ne pouvoir ouvrir que les images (.jpeg) et donc éviter les erreurs d'ouverture. La deuxième est un nouvel évènement permettant de déclencher le programme d'exécution de résolution du sudoku à l'appui de la touche "s".

6.4.3 Résultat

En assemblant tout cela, on obtient une application avec une interface simple et facile d'utilisation permettant de charger un sudoku et l'afficher résolu sur la fenêtre.

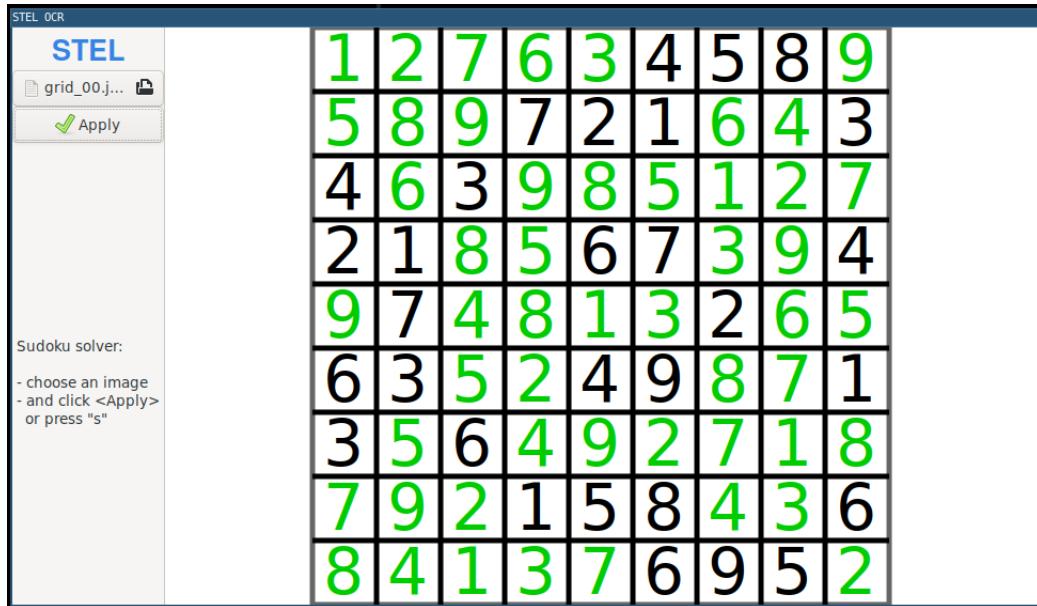


Figure 39: L'application finale

7 Conclusion

C'est la seconde fois que nous travaillons en groupe sur un projet avec un but bien défini. De l'avis général, nous avons consolidé nos connaissances générales et appris à faire des applications plus attrayantes et plus orientées pour le monde du travail. Nous sommes globalement satisfaits de ce que nous avons réalisé.

Au niveau de la gestion du projet en équipe, nous avons réussi à bien nous répartir les tâches afin de réaliser nos objectifs dans les temps et l'ambiance générale du groupe était très bonne.

Une bonne expérience à renouveler !