

Epita 2022 –Rapport de soutenance- Projet S3

Yanis BAYLE
Pierre LAVIELLE
Erwan MAIGNE-MONTAMAT

OCR SUDOKU

Site web: <https://ocrsudokuepitas3.000webhostapp.com/>

Table des matières:

- I. Introduction:
 - A. présentation du groupe
 - B. plannings des tâches
 - C. problèmes rencontrés
- II. Aspect techniques
 - A. réseau de neurones
 - B. prétraitement
 - C. affichage
 - D. Accessibilité du code
- III. Conclusion

I. Introduction

A. présentation de l'équipe

Yanis BAYLE:

Ce projet m'a permis d'en apprendre plus sur le travail en équipe et comment fonctionner en groupe afin de mener à bien la réalisation d'un projet à première vue très complexe. J'ai pu dans ce projet mettre en pratique la notion d'allocation dynamique de la mémoire spécifique au langage C, une notion que nous n'avions seulement abordée en cours auparavant. De plus, j'ai pu découvrir ce qu'était un OCR ainsi que son mode de fonctionnement, et j'ai pu me rendre compte à quel point cette technologie était présente autour de nous.

La nécessité d'utiliser un réseau de neurones afin de résoudre le problème de la reconnaissance de caractères manuscrits à partir d'une image, m'a permis d'obtenir une nouvelle façon de résoudre un problème complexe qu'il serait impossible de résoudre par des méthodes de programmation plus basiques. De ce fait, j'ai pu découvrir l'existence des réseaux de neurones qui permettent aux programmes de reconnaître des modèles et de résoudre des problèmes courants dans les domaines de l'IA, de l'apprentissage automatique et de l'apprentissage profond.

Pierre LAVIELLE:

De nombreux obstacles nous sont arrivés tel que nous n'étions que trois, que nous n'avions aucune idée de comment commencer ce projet et comparée à l'année où nous apprenions ce langage, le SDL que nous utilisons a été appris seul et de manière indépendante.

Ce projet m'a permis d'apprendre le langage SDL et glade tout en découvrant comment traiter une image en détail. Nous avons créé une bonne ambiance et donc par la même occasion une bonne cohésion de groupe, ce qui nous a permis de travailler dans des conditions plus que favorables afin de pouvoir avancer dans notre projet sans trop d'encombres.

Ceci tout en restant dans les délais imposés sans être débordés par le travail demandé. Pour ma part, je devais m'occuper de retirer les couleurs de l'image pour avoir une image entièrement noire et blanche, j'ai donc appris à me servir du langage C avec l'aide de mes collègues quand je me retrouvais bloquée et que je n'arrivais pas à trouver le problème. Sans avoir un groupe aussi soudé que celui-ci cela aurait été impossible.

Erwan MAIGNE-MONTAMAT:

Cette année étant redoublant c'est la seconde fois que je suis confronté à cet exercice, la partie prétraitement ayant été relativement bien réussie l'année dernière (bien que largement perfectible) la deuxième soutenance ne sera problématique que dans les notions finales du prétraitement tel que la détection de la grille la rotation de cette dernière et son découpage qui ont été finis dans la précipitation. La problématique principale ayant été la communication avec l'un des membres du groupe ayant fait le mort, et le fait de ne pas se faire submerger par la charge de travail, ayant eu des côtés très énergivore et chronophage.

B. planning des tâches

<i>Charges</i>	B. Yannis	L. Pierre	M. Erwan
<i>Résolution du sudoku</i>	X		
<i>Chargement de l'image</i>	X	X	
<i>Suppression des couleurs</i>		X	X
<i>Rotation manuelle de l'image</i>			X
<i>Détection de la grille</i>	X		X
<i>Découpage de l'image</i>		X	X
<i>Réseau de neurones (XOR)</i>	X	X	

II. Aspect technique

A. réseaux de neurones

Reconnaissance de caractères (réseau de neurones):

La reconnaissance de caractères est une partie centrale de tout OCR. Un réseau de neurones combiné à de l'apprentissage profond fournissent actuellement les meilleures solutions à de nombreux problèmes de reconnaissance d'images, d'où son utilisation pour ce projet.

Un réseau de neurones est constitué d'une couche d'entrée, c'est-à-dire l'ensemble de neurones qui portent le signal d'entrée, d'une ou plusieurs couches cachées, qui constituent le cœur de notre réseau et est là où les relations entre les variables vont être mises en et d'une couche de sortie qui représente le résultat final de notre réseau.

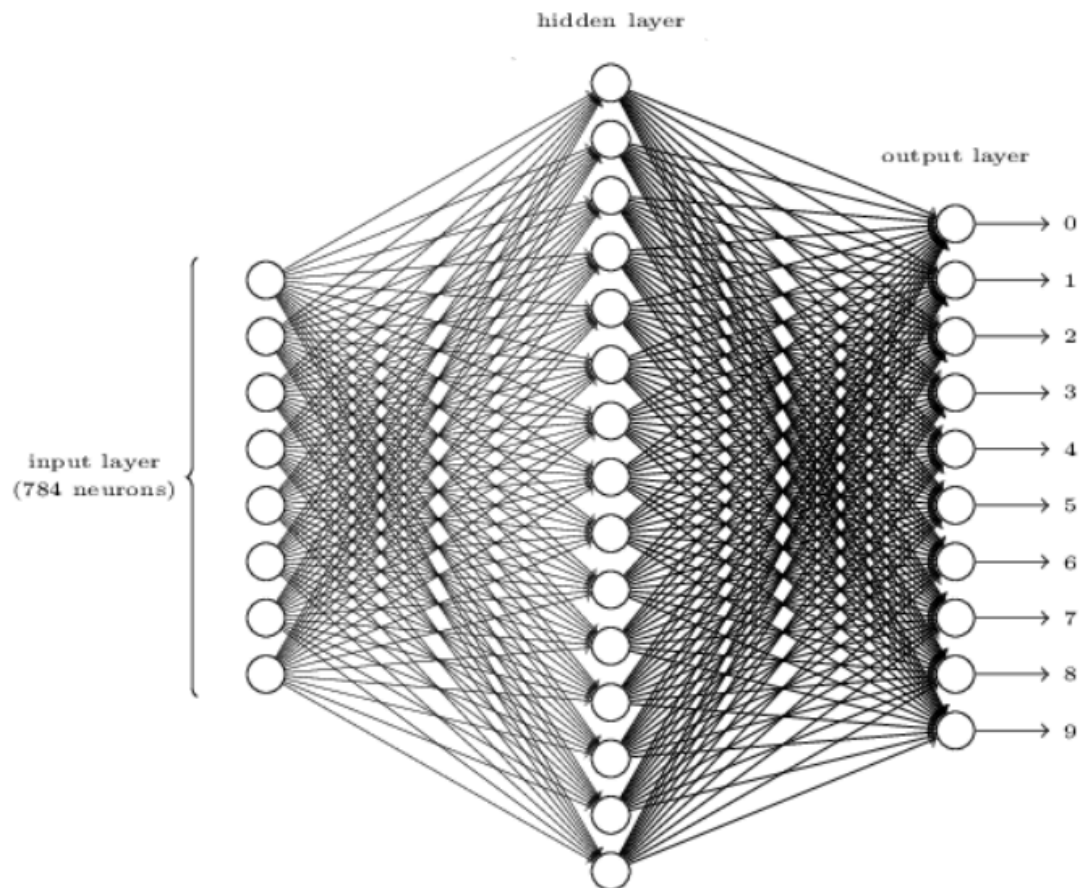
Notre réseau de neurones est un réseau de neurones "feedforward". Comme son nom l'indique, les données d'entrée sont transmises dans le sens direct à travers le réseau. Chaque couche cachée accepte les données d'entrée, les traite selon la fonction d'activation et les transmet à la couche suivante.

Cela signifie qu'il n'y a pas de boucles dans le réseau - les informations sont toujours transmises, jamais renvoyées. Si nous avions des boucles, nous nous retrouverions dans des situations où l'entrée de la fonction dépendrait de la sortie. Ce n'est pas le cas

dans la reconnaissance de caractère, c'est pourquoi nous n'utilisons pas de telles boucles.

Afin d'implémenter ce type de réseau en C, nous créons d'abord une nouvelle structure afin de représenter notre structure de neurones. Cette structure contient une liste d'inputs (entrées), une liste de weights (poids), et une valeur de biais.

Nous utiliserons ces neurones afin de constituer 3 couches distinctes dans notre réseau de neurones, une couche d'entrée (input layer), une couche cachée (hidden layer), et une couche de sortie (output layer). Notre réseau de neurones peut alors s'apparenter au schéma ci-dessous.



Explicitons le rôle de chacune de ces couches et leur fonctionnement.

La couche d'entrée:

La couche d'entrée du réseau contient des neurones codant les valeurs des pixels d'entrée. Nos données d'entraînement pour le réseau seront constituées de nombreuses images de 28 par 28 pixels de chiffres manuscrits scannés, et la couche d'entrée contient donc 784 (28×28) neurones. Les pixels d'entrée sont en noir ou en blanc, avec une valeur rgb de 255,255,255 représentant le blanc, et une valeur rgb de 0,0,0 représentant le noir. Ces images sont obtenus à la suite du prétraitement et du découpage d'une image d'une grille de sudoku contenant des chiffres écrits à la main, cependant afin d'obtenir le maximum de ce type d'image,

nous avons notamment utilisé la base de données MNIST de chiffres manuscrits.

Cette base de données comporte un ensemble d'apprentissage de 60 000 exemples et un ensemble de tests de 10 000 exemples. Il s'agit d'un sous-ensemble d'un ensemble plus important disponible auprès du NIST. Les chiffres ont été normalisés en taille et centrés dans une image de taille fixe.

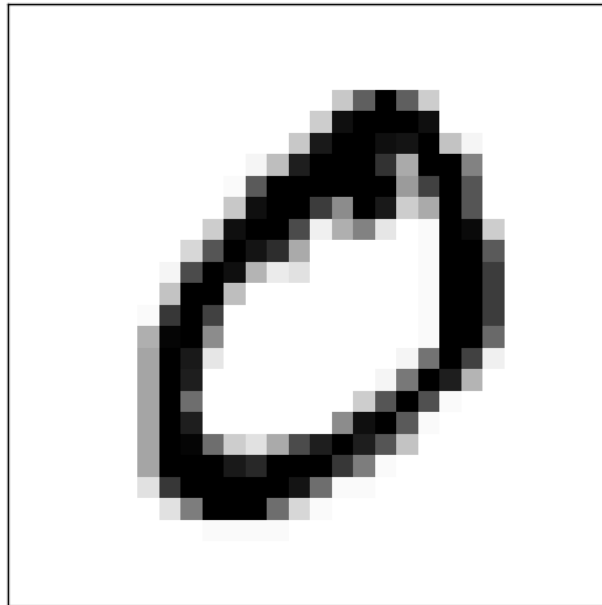
Il s'agit d'une bonne base de données pour des techniques d'apprentissage et des méthodes de reconnaissance des formes sur des données du monde réel tout en consacrant un minimum d'efforts au prétraitement et au formatage.



(exemple d'une partie des chiffres contenues dans cette base de données)

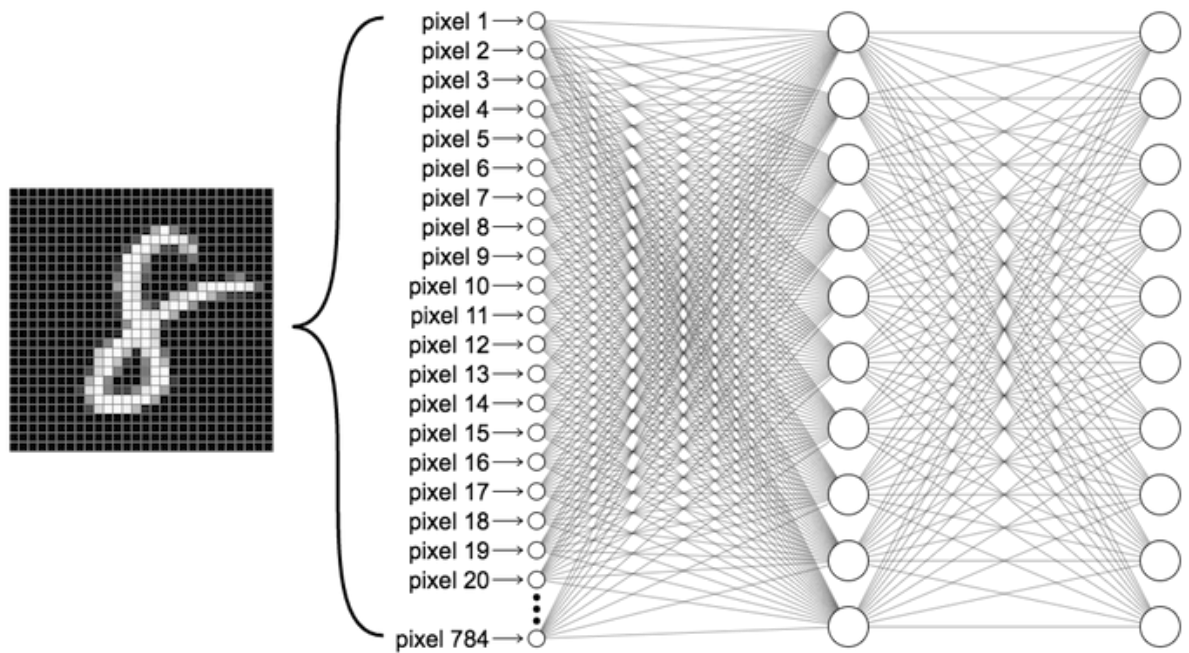
Évidemment l'utilisation de cette base de données est utilisée purement pour faciliter l'entraînement du réseau de neurone et des données de situations réels obtenus depuis l'image d'un sudoku seront utilisées à la suite de cette phase.

Cette base de données est découpé en carrés de 28*28 contenant un nombre manuscrit sur lesquels nous allons entraîner notre réseau de neurones.



(exemple du type d'input attendu)

A partir de ce type d'image, notre programme va parcourir chaque pixel de l'image ligne par ligne jusqu'à avoir parcouru chaque pixel de l'image. Chaque pixel rencontré de cette façon est donné à un des neurones d'entrée du réseau de neurone. Le réseau de neurones peut alors essayer de déterminer quel est ce nombre.



(exemple de données donné à la couche d'entrée)

La couche intermédiaire :

Notre couche intermédiaire n'est pas constituée de perceptrons, comme pour un réseau de neurones simples, mais de neurones sigmoïdes. Les neurones sigmoïdes sont similaires aux perceptrons, mais ils sont légèrement modifiés de telle sorte que la sortie du neurone sigmoïde est beaucoup plus lisse que la sortie fonctionnelle en escalier du perceptron. Cela permet à notre réseau de neurones de pouvoir effectuer ses calculs avec plus de précision.

En effet, un perceptron n entrées (x_1, x_2, \dots, x_n) et à une seule sortie o est défini par la donnée de n poids (ou coefficients synaptiques) (w_1, w_2, \dots, w_n) et un biais (ou seuil) θ par:

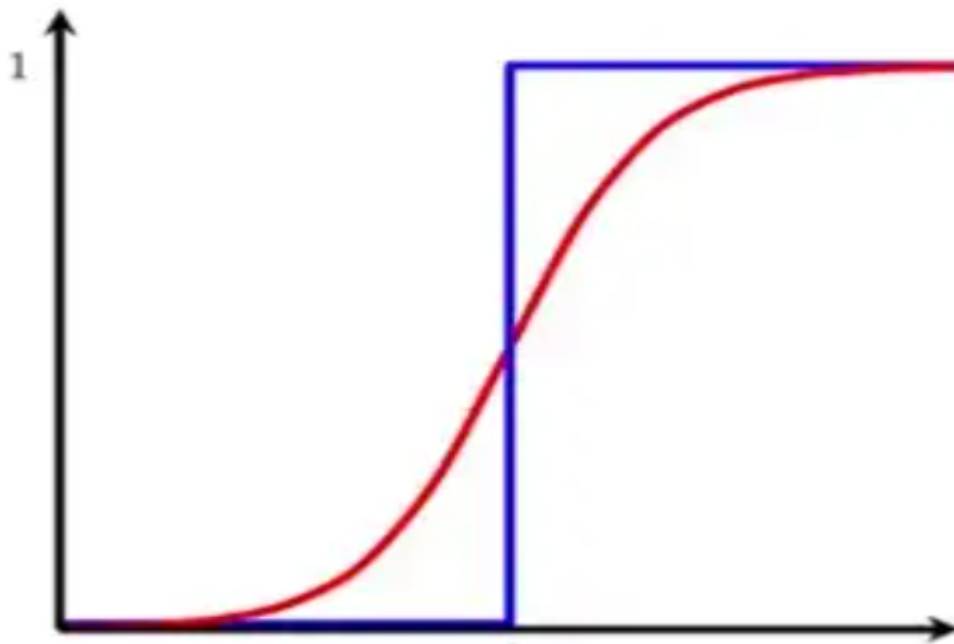
$$o = f(z) = \begin{cases} 1 & \text{si } \sum_{i=1}^n w_i x_i > \theta \\ 0 & \text{sinon} \end{cases}$$

D'après la représentation mathématique, nous pourrions dire que la logique de seuillage utilisée par le perceptron est très dure. Voyons cette logique de seuillage sévère à l'aide d'un exemple.

Considérons le processus de décision du neurone, dans le cas où le résultat de la somme définissant le perceptron est égale à la valeur de seuil, le neurone ne se déclenche pas, mais en revanche si cette somme est ne serait-ce plus grande que la valeur de seuil de 1 millième le neurone se déclenche.

Ce comportement n'est pas une caractéristique du problème spécifique que nous choisissons ou du poids spécifique et du seuil que nous choisissons. Il s'agit d'une caractéristique du neurone du perceptron lui-même qui se comporte comme une fonction à pas. Nous pouvons surmonter ce problème en introduisant un nouveau type de neurone artificiel appelé neurone sigmoïde.

Les neurones sigmoïdes ont la fonction de sortie est beaucoup plus lisse que la fonction en escalier. Dans le neurone sigmoïde, une petite variation de l'entrée ne provoque qu'une petite variation de la sortie, contrairement à la sortie en escalier des perceptrons.



(comparaison des résultats de sortie possible d'un perceptron (bleu) et de neurone sigmoïde)

Nous ne voyons plus de transition nette à la valeur seuil. La sortie du neurone sigmoïde n'est pas 0 ou 1. Il s'agit plutôt d'une valeur réelle comprise entre 0 et 1 qui peut être interprétée comme une probabilité. Cela nous permet d'avoir un réseau de neurones plus précis et donc plus performant.

Elle est couramment utilisée pour les modèles où nous devons prédire la probabilité en tant que sortie. Comme la probabilité d'une chose n'existe qu'entre 0 et 1, la fonction sigmoïde est le bon choix en raison de son étendue.

La fonction est différentiable et fournit un gradient lisse, c'est-à-dire qu'elle empêche les sauts dans les valeurs de sortie. Ceci est représenté par une forme en S de la fonction d'activation sigmoïde.

La fonction sigmoïde des neurones est définie pour tout réel x par :

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

Dans notre cas x sera égale à la somme du produit des neurones d'entrée par leur poids moins le biais.

Sa dérivée (que nous utiliserons plus tard) quant à elle est définie pour tout réel x par :

$$f'(x) = f(x) \cdot (1 - f(x))$$

La couche de sortie:

La couche de sortie du réseau contient 10 neurones. Si le premier neurone se déclenche, c'est-à-dire qu'il a une sortie ≈ 1 , alors cela indiquera que le réseau pense que le chiffre est un 0. Si le deuxième neurone se déclenche alors cela indiquera que le réseau pense que le chiffre est un 1. Et ainsi de suite. De manière un peu plus précise, nous numérotons les neurones de sortie de 0 à 9, et nous déterminons quel neurone a la valeur d'activation la plus élevée. Si ce neurone est, disons, le neurone numéro 6, alors notre réseau supposera que le chiffre d'entrée était un 6. Et ainsi de suite pour les autres neurones de sortie.

Nous avons au départ opté pour seulement 4 neurones de sortie en traitant chaque neurone comme prenant une valeur binaire, selon que la sortie du neurone est plus proche de 0 ou de 1. Quatre neurones suffisent pour coder la réponse, puisque 2^4 est plus que les 10 valeurs possibles pour le chiffre d'entrée.

Cependant, selon nos tests effectués avec ces deux versions du réseau de neurone, il s'avère que, pour ce problème particulier, le réseau avec 10 neurones de sortie apprend à reconnaître les chiffres mieux que le réseau avec 4 neurones de sortie, d'où notre choix d'utiliser 10 neurones de sortie malgré que cela semble contre intuitif.

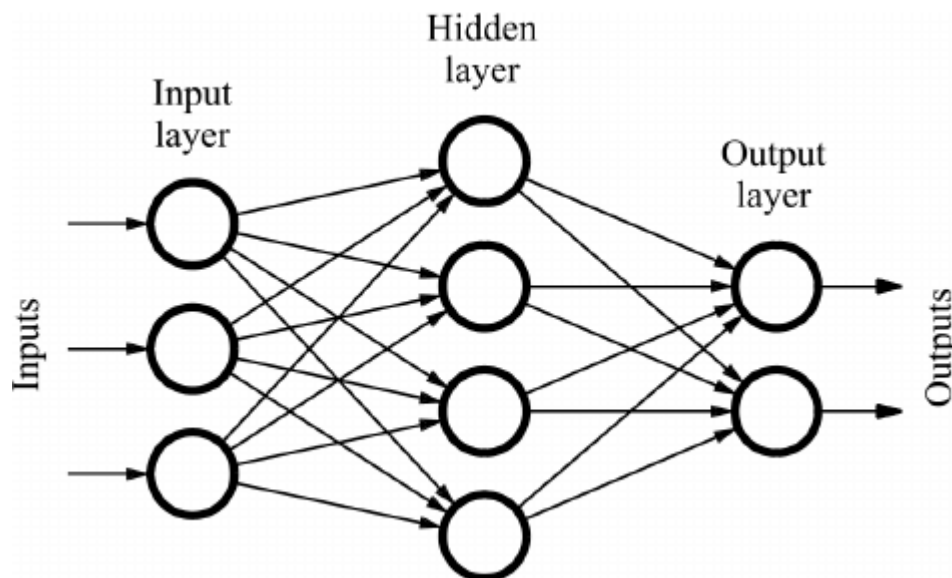
Maintenant que nous avons conçu notre réseau neuronal, il doit maintenant apprendre à reconnaître les chiffres. La première chose dont nous avons besoin, c'est d'un ensemble de données pour l'apprentissage - un ensemble de données d'entraînement. Nous utiliserons le jeu de données MNIST (comme annoncé précédemment) , qui contient des dizaines de milliers d'images scannées de chiffres manuscrits, ainsi que leurs classifications correctes.

Avant de lancer le programme, les poids et le biais se voient attribuer une valeur aléatoire comprise entre 0 et 1. La valeur initiale des poids et du biais importent peu étant donné qu'elles seront amenées à être modifiées pendant la phase d'entraînement, afin d'atteindre des valeurs permettant au réseau de neurone de reconnaître des chiffres manuscrits avec le maximum de précision.

La phase d'entraînement se fait en 3 étapes:

Le programme va d'abord calculer le résultat obtenu avec les poids et le biais actuel. Chaque pixel de l'image de départ (image de 28*28 pixels en noir et blanc d'un chiffre manuscrit) est donné à

un neurone d'entrée de la couche d'entré (chaque neurone de la couche d'entré à toujours un seul pixel en entré). Ces données sont envoyées aux neurones de la couche cachée qui vont alors utiliser la fonction sigmoïde décrite plus tôt afin de déterminer leur sortie. La couche cachée va alors envoyer les données obtenues à la couche de sortie. Un des 10 neurones de la couche de sortie (celui qui aura le résultat le plus élevé) donnera le résultat de l'opération. Cette étape s'appel "forward propagation" (propagation vers l'avant en français), elle peut être représenté par le schéma ci-dessous:

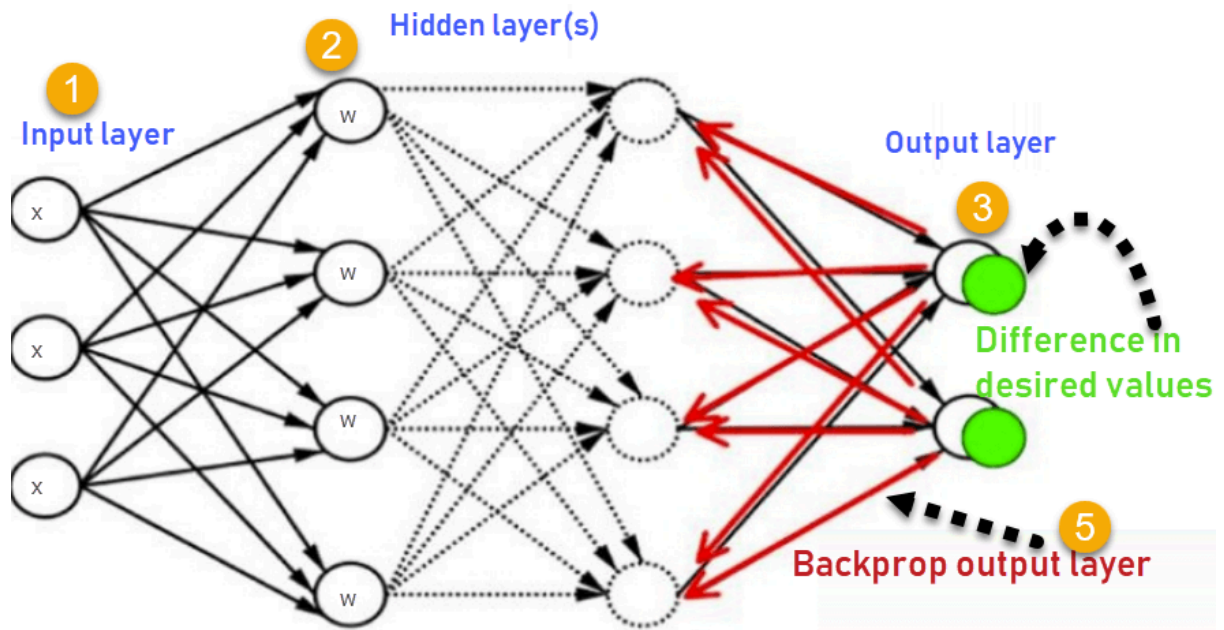


(représentation du principe de la propagation par l'avant d'un réseau de neurones)

La seconde étape consiste à calculer l'erreur entre le résultat que l'on a trouvé et le résultat attendu. La valeur de l'erreur est égale à la dérivé fonction sigmoïde décrite plus tôt multiplié par la soustraction du résultat attendu par le résultat obtenu.

Enfin, il reste à mettre à jour le poids et le biais des neurones de notre réseau de neurones en fonction de l'erreur que nous avons calculée à l'étape précédente. Cette étape s'appelle "backward propagation" (propagation vers l'arrière) peut être

représenté par le schéma suivant :



Ces 3 étapes seront répétées un certain nombre de fois (dans notre cas nous avons choisi 4000 fois car cela permet un bon compromis entre précision et vitesse d'exécution). A la suite de cela notre réseau de neurones sera suffisamment entraîné pour reconnaître des chiffres manuscrits avec un taux de réussite satisfaisant.

B. Le prétraitement:

Le prétraitement est composé d'une succession d'étape que nous allons vous détailler ici:

- Le chargement de l'image
- Sa binarisation:
 - La réduire à des nuances de gris
 - La réduire à deux nuances: Noir et blanc
 - Supprimer le bruit
- Trouver la grille
- Utiliser la grille pour déduire l'angle nécessaire pour obtenir une image exploitable
- Effectuer la rotation de l'image et isoler la grille
- Découper la grille en 81 images séparées et les rendre

Il est à noter qu'après le passage dans l'OCR il faut récupérer les réponses renvoyées par ce dernier afin de reconstituer la grille et l'envoyer dans le solveur de sudoku.

Le chargement de l'image:

Le chargement de l'image s'effectue simplement avec la librairie SDL (librairie disponible sur les ordinateurs d'Epita):

```
image_surface = load_image("my_image.jpg");
```

La binarisation d'une image:

Une fois l'image chargée il y a une série de manipulation à effectuer afin de la rendre exploitable:

Tout d'abord nous la réduisons à des nuances de gris comme avec les TP effectués en début d'année

```
void grayscale(SDL_Surface *surface, int width, int height)
{
    for(int x = 0; x < width; x++)
    {
        for(int y = 0; y < height; y++)
        {
            Uint32 pixel = get_pixel(surface, x, y);
            Uint8 r, g, b;
            SDL_GetRGB(pixel, surface->format, &r, &g, &b);
            Uint8 average = 0.212671 * r + 0.715160 * g + 0.072169 * b;
            // Uint8 average = 0.3*r + 0.59*g + 0.11*b;
            r = g = b = average;
            Uint32 new_color = SDL_MapRGB(surface->format, r, g, b);
            put_pixel(surface, x, y, new_color);
        }
    }
}
```

Enfin si le gris obtenu est d'une valeur supérieur à (128,128,128) on considérera que le pixel est blanc, et inversement si il est inférieur à cette valeur on le considérera noir.

Enfin la prochaine et dernière étape de la binarisation est de supprimer le bruit avec au moment de l'écriture de ces lignes un algorithme qui fait une moyenne de la couleur du dit pixel et des pixels adjacents, il est important de noter que le débruitage par patch aura probablement également été ajouté au moment où vous verrez la présentation.

Vous trouverez ci dessous un exemple de la réduction de bruit utilisée à l'heure de l'écriture de ces lignes.

```
using a median filter using a 3x3 window
DL_Surface *NoiseReduction(DL_Surface *image)

    // image = SDL_ConvertSurface(image, SDL_PIXELFORMAT_ARGB8888, 0);

    Uint32 pixel;

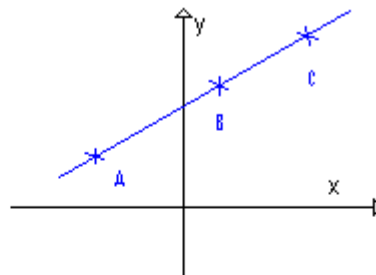
    int window[9];

    for (int j = 0; j < image->h; ++j)
    {
        for (int i = 0; i < image->w; ++i)
        {
            // Border cases
            if (i==0)
            {
                if (j == 0)
                {
                    window[0] = get_pixel(image, i, j);
                    window[1] = get_pixel(image, i, j);
                    window[2] = get_pixel(image, i, j+1);
                    window[3] = get_pixel(image, i, j);
                    window[4] = get_pixel(image, i, j);
                    window[5] = get_pixel(image, i, j+1);
                    window[6] = get_pixel(image, i+1, j);
                    window[7] = get_pixel(image, i+1, j);
                    window[8] = get_pixel(image, i+1, j+1);
                }
                else if (j==(image->h-1))
                {
                    window[0] = get_pixel(image, i, j-1);
                    window[1] = get_pixel(image, i, j);
                    window[2] = get_pixel(image, i, j);
                    window[3] = get_pixel(image, i, j-1);
                    window[4] = get_pixel(image, i, j);
                    window[5] = get_pixel(image, i, j);
                    window[6] = get_pixel(image, i+1, j-1);
                    window[7] = get_pixel(image, i+1, j);
                    window[8] = get_pixel(image, i+1, j);
                }
                else
                {
                    window[0] = get_pixel(image, i, j-1);
                    window[1] = get_pixel(image, i, j);
                    window[2] = get_pixel(image, i, j+1);
                    window[3] = get_pixel(image, i, j-1);
                    window[4] = get_pixel(image, i, j);
                    window[5] = get_pixel(image, i, j+1);
                    window[6] = get_pixel(image, i+1, j-1);
                    window[7] = get_pixel(image, i+1, j);
                    window[8] = get_pixel(image, i+1, j+1);
                }
            }
        }
    }
}
```

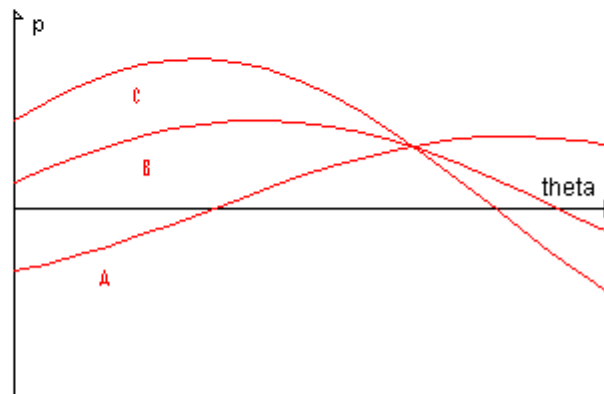
Trouver la grille:

Pour ce faire nous utilisons l'algorithme de Hough aussi appelé transformation de Hough, ce dernier va prendre l'ensemble des lignes pouvant passer par un point, puis tracer la perpendiculaire à cette droite telle qu'elle passe par l'origine, cette distance à l'origine nous donnera la distance p , et θ l'angle de la dite droite, on pourra donc en conclure l'existence d'une droite.

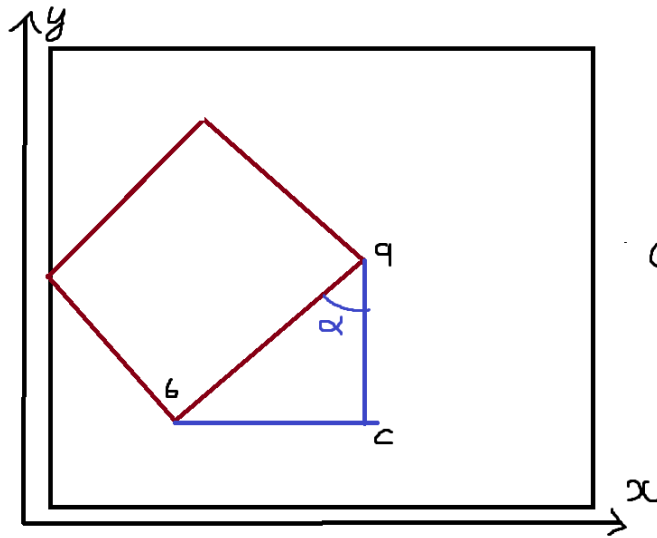
Image originale



Espace de Hough



Une fois l'existence de ladite droite confirmée, il devient facile de déterminer l'angle nécessaire pour la rotation.



$$\begin{aligned} a &= (x_a, y_a) & \text{CAH} \\ b &= (x_b, y_b) & \text{SOH} \\ c &= (x_a, y_b) & \text{TOA} \end{aligned}$$

$$\cos(\alpha) = \frac{(y_a - y_c)}{\sqrt{|AB|^2}}$$

Ceci se traduit de la manière suivante une fois codé:

```
int HoughTransform(SDL_Surface* image_surface)
{
    int w = (image_surface->w);
    int h = (image_surface->h);

    int roCompress = 10;
    size_t diagonale = sqrt(w * w + h * h);
    unsigned int mat[diagonale][90];
    int ro;
    diagonale /= roCompress;
    float pi = 3.1415926535897;
    Uint8 r, g, b;
    Uint32 pixel;

    for (size_t i = 0; i < diagonale; i++)
    {
        for (int j = 0; j < 90; j++)
        {
            mat[i][j] = 0;
        }
    }

    for (int x = 0; x < w; x++)
    {
        for (int y = 0; y < h; y++)
        {
            for (int taita = -45; taita < 45; taita++)
            {
                float radTaita = (float)taita * (pi / 180);
                ro = round(x * cos(radTaita) + y * sin(radTaita));
                ro /= roCompress;

                pixel = get_pixel(image_surface, x, y);
                SDL_GetRGB(pixel, image_surface->format, &r, &g, &b);

                if (r == 255 && g == 255 && b == 255 && ro < (int)diagonale)
                {
                    mat[ro][taita] += 1;
                }
            }
        }
    }
}
```

Enfin il est possible alors de déduire toute la grille en parcourant ligne par ligne et colonne par colonne pour chercher l'existence de la grille, ou d'utiliser Hough pour tout déduire, à l'écriture de ces lignes nous privilégions la première option. Une fois ceci fait nous pouvons créer une nouvelle image de la taille de la grille uniquement dans le bon angle et attaquer la suite du découpage.

Enfin pour la création des sous images il suffit de couper la grille en fonction de leur coordonnée en 81 petites images, cette méthode laisse souvent des parties des jonctions des cases en trop sur les côtés, mais nous laissons à l'OCR le devoir de ne pas les qualifier de pertinentes.

Voici donc un exemple de la manière idéale de procéder:

Notre image en sorti de la rotation automatique:

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

Nous prenons la liberté de la redimensionner à une taille fixe de 900 x 900 pixel afin d'avoir un nombre d'input défini pour le réseau de neurones:

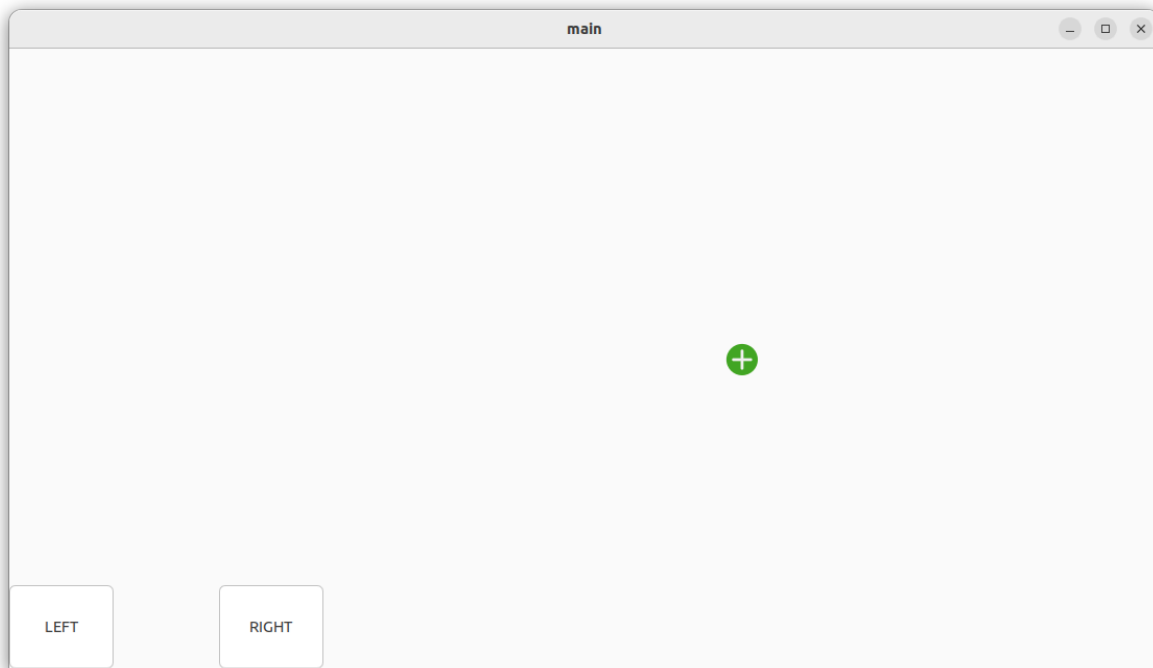
5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Puis nous découpons tous les 100 pixels ladite image (vous pouvez voir dans le quadrillage ci contre la taille initiale de l'image et générons une nouvelle image pour chaque case.

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

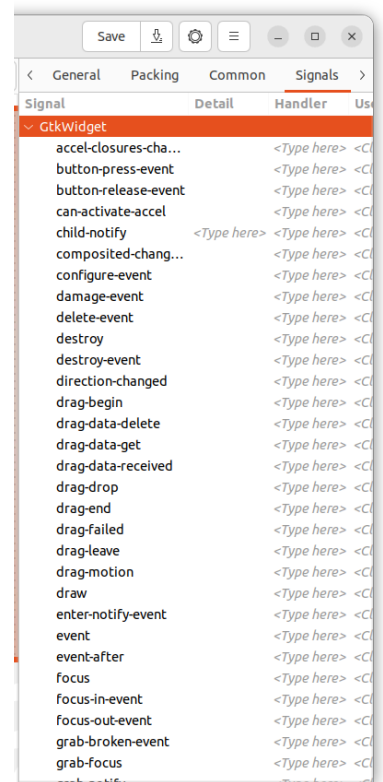
C. Affichage:

Pour l'affichage de notre projet nous avons décidé d'utiliser glade et la librairie gtk.



glade permet de créer des interfaces utilisateurs avec une aisance plus qu'appréciable, il permet de rapidement connecter un objet a une fonction et de pouvoir ainsi l'appeler de l'interface sans avoir besoin d'écrire des lignes de code.

Les manières de modifier un objet sont nombreuses: on peut le déplacer, le réorganiser, l'activer ... et glade permet d'appeler une fonction spécifique que nous avons créé pour chaque type de modification utilisé sur un objet.

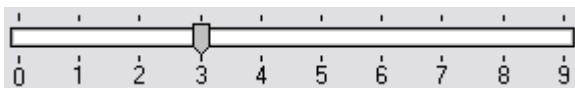


sur cette fenêtre vous pouvez voir une zone de “drag and drop” et deux boutons :

- la zone de “drag and drop” permet à l'utilisateur de pouvoir facilement déposer son image dans glade et par le découpage d'image présenter l'image de manière quadriller.
- quand les boutons sont activés ils sont supposés tourner l'image de 90 degrés soit à droite soit à gauche.

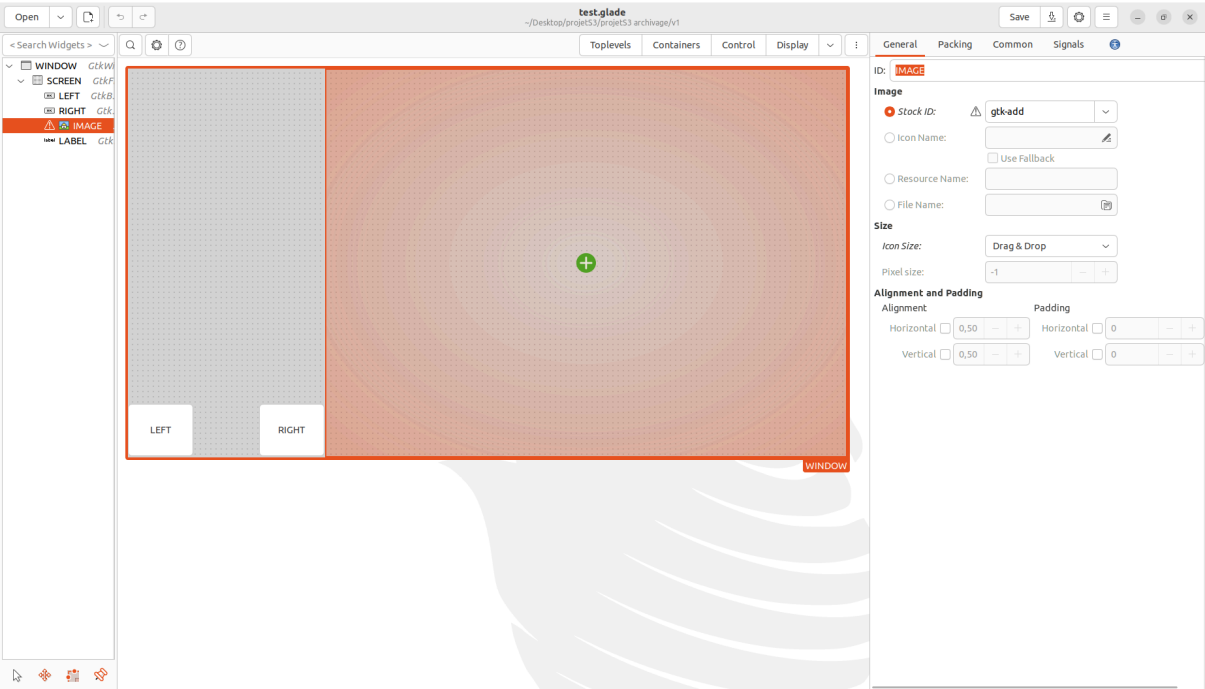
Les connections entre les fonctions et les objets ont été perturbées ainsi elles peuvent faire un effet mais pour le montrer nous avons fait apparaître un texte à la place.

Le produit fini de cette interface prévoyait de changer nos boutons en glisseur pour pouvoir mieux tourner l'image,



de rajouter des boutons permettant de réécrire sur l'image et d'autres fonctionnalités.

Nous pensions aussi rajouter d'autres spécificités à notre ocr comme la possibilité de rajouter un chiffre pour aider l'utilisateur au lieu de résoudre directement le sudoku cependant nous avons choisis de nous concentrer sur les problèmes encourus sur notre projet et pouvoir l'optimiser.



D. Accessibilité du code:

Une fois extrait l'arborescence du projet est la suivante, afin d'avoir accès à la dernière version fonctionnelle du code merci de suivre le chemin suivant:

PC > DATA (D:) > TPepita > projet s3bis > projetS3 >

Nom	Modifié le	Type	Taille
.git	16/12/2022 21:50	Dossier de fichiers	
image	16/12/2022 21:43	Dossier de fichiers	
jeu de données	16/12/2022 13:46	Dossier de fichiers	
NEURAL NETWORK OCR (marche po wall...	16/12/2022 21:43	Dossier de fichiers	
prétraitement	17/12/2022 20:21	Dossier de fichiers	
projetS3 archivage	16/12/2022 21:50	Dossier de fichiers	
rapport de soutenances	16/12/2022 13:46	Dossier de fichiers	
XOR NEURAL	16/12/2022 21:43	Dossier de fichiers	
AUTHORS	16/12/2022 21:43	Fichier	1 Ko
image_01.jpeg	16/12/2022 21:43	Fichier JPEG	98 Ko
main.c	16/12/2022 21:48	C Source	2 Ko
make2	16/12/2022 21:43	Fichier	1 Ko
Makefile	16/12/2022 21:43	Fichier	1 Ko
README	16/12/2022 21:43	Fichier	1 Ko
solver.c	16/12/2022 21:43	C Source	8 Ko
test.glade	16/12/2022 21:43	Fichier GLADE	4 Ko

Projet S3/Projet s3archivage/v1

PC > DATA (D:) > TPepita > projet s3bis > projetS3 >

Nom	Modifié le	Type	Taille
.git	16/12/2022 21:50	Dossier de fichiers	
image	16/12/2022 21:43	Dossier de fichiers	
jeu de données	16/12/2022 13:46	Dossier de fichiers	
NEURAL NETWORK OCR (marche po wall...	16/12/2022 21:43	Dossier de fichiers	
prétraitement	17/12/2022 20:21	Dossier de fichiers	
projetS3 archivage	16/12/2022 21:50	Dossier de fichiers	
rapport de soutenances	16/12/2022 13:46	Dossier de fichiers	
XOR NEURAL	16/12/2022 21:43	Dossier de fichiers	
AUTHORS	16/12/2022 21:43	Fichier	1 Ko
image_01.jpeg	16/12/2022 21:43	Fichier JPEG	98 Ko
main.c	16/12/2022 21:48	C Source	2 Ko
make2	16/12/2022 21:43	Fichier	1 Ko
Makefile	16/12/2022 21:43	Fichier	1 Ko
README	16/12/2022 21:43	Fichier	1 Ko
solver.c	16/12/2022 21:43	C Source	8 Ko
test.glade	16/12/2022 21:43	Fichier GLADE	4 Ko

Ici vous verrez notre jeu de donnée et les bases de la liaison entre les différentes partie du projet mais ce qui nous intéressera car déjà fonctionnel est le prétraitement:

'C > DATA (D:) > TPepita > projet s3bis > projetS3 > projetS3 archivage > v1

Nom	Modifié le	Type	Taille
image	16/12/2022 21:50	Dossier de fichiers	
jeu de données	16/12/2022 21:50	Dossier de fichiers	
NEURAL NETWORK OCR (marche po wall...	16/12/2022 21:50	Dossier de fichiers	
prétraitement	16/12/2022 21:50	Dossier de fichiers	
XOR NEURAL	16/12/2022 21:50	Dossier de fichiers	
AUTHORS	22/10/2022 18:20	Fichier	1 Ko
main.c	22/10/2022 18:20	C Source	2 Ko
Makefile	22/10/2022 18:20	Fichier	1 Ko
README	22/10/2022 18:20	Fichier	1 Ko
solver.c	30/10/2022 17:54	C Source	8 Ko
test.glade	22/10/2022 18:20	Fichier GLADE	4 Ko





PC > DATA (D:) > TPepita > projet s3bis > projetS3 > projetS3 archivage > v1 > prétraitement >

Nom	Modifié le	Type	Taille
.vscode	16/12/2022 21:50	Dossier de fichiers	
Utile powerpoint présentation	16/12/2022 21:50	Dossier de fichiers	
grayscale.c	23/10/2021 18:03	C Source	11 Ko
grayscale.h	23/10/2021 17:51	C/C++ Header	1 Ko
hough.c	17/12/2022 18:47	C Source	4 Ko
hough.h	17/12/2022 18:47	C/C++ Header	1 Ko
main.c	17/12/2022 18:47	C Source	2 Ko
Makefile	17/12/2022 18:47	Fichier	1 Ko
my_image.jpg	04/09/2021 21:19	Fichier JPG	680 Ko
my_image_good.jpg	04/09/2021 21:18	Fichier JPG	98 Ko
my_image_m.jpg	14/10/2021 15:23	Fichier JPG	10 Ko
my_image_white.jpg	04/09/2021 21:19	Fichier JPG	258 Ko
my_imaged.jpg	04/09/2021 21:18	Fichier JPG	154 Ko
pixel_operations.c	14/10/2021 15:23	C Source	2 Ko
pixel_operations.h	14/10/2021 15:23	C/C++ Header	1 Ko
remove_color.c	14/10/2021 15:23	C Source	3 Ko
remove_color.h	14/10/2021 15:23	C/C++ Header	1 Ko
solver.c	17/12/2022 18:47	C Source	4 Ko
tools.c	21/10/2021 16:23	C Source	6 Ko
tools.h	21/10/2021 16:22	C/C++ Header	1 Ko

Il est à noter que l'ocr marche théoriquement mais nous n'avons pas eu le temps de l'implémenter ou l'entraîner.

e ATTICNAGE

A (D:) > TPepita > projet s3bis > projetS3 > projetS3 archivage > v1 > NEURAL NETWORK OCR (marche po wallah mais on a try)

Nom	Modifié le	Type	Taille
 .gitignore	17/04/2017 13:01	Document texte	1 Ko
 Makefile	17/04/2017 13:01	Fichier	1 Ko
 neural_network.c	17/12/2022 19:28	C Source	9 Ko
 neural_network.h	17/12/2022 19:28	C/C++ Header	2 Ko

III. Conclusion

En conclusion nous avons fait de notre mieux pour pouvoir pallier notre manque de main d'œuvre cependant ce manque a été plus que dévastateur pour notre projet. l'absence d'un quatrième membre nous a obligés de parfois travailler seul sur une partie conséquente de l'OCR.

Au début nous avons essayé de séparer les tâches de chacun afin de travailler plus efficacement cependant sur la fin du projet les retards des uns bloquant les autres nous nous sommes retrouvés très limités. Un contrôle plus régulier des avancements aurait peut être été plus bénéfique pour la réalisation du projet.