Première Soutenance - Projet OCR

Groupe 5

Octobre 2021

Guy Noe DJOUFAING NZUMGUENG

Emre ULUSOY

Thomas LEGRAND

Youness BAKILY

Table des matières

1 Introduction

Cette année, de Septembre à fin Octobre s'est déroulé la première partie du projet d'Info Spé nécessitant de réaliser les pré requis d'un OCR (Optical Character Recognition en anglais) utilisé pour résoudre un sudoku. Notre groupe , composé de Guy Noe DJOUFAING NZUMGUENG en tant que chef de groupe ainsi que de Emre ULUSOY, Youness BAKILY et Thomas LEGRAND.

2 Organisation du projet

Pour pouvoir garder le projet organisé et être capable de respecter les deadlines du cahier des charges, nous devons attribué à chaque membre du projet, les tâches figurant sur le cahier des charges.

2.1 Ressources utilisées

Pour ce projet, nous utilisons des logiciels open-source et des outils mis à disposition par Epita tel que :

— Overleaf (Un éditeur LaTeX)

— Sublime-Text 3, VSCode et VIM

— Git et GitKraken

— Le compilateur GCC

2.2 Répartition des charges

Younes :

Younes était chargé de s'occuper de la partie de la rotation manuelle de l'image pour cette première soutenance.

Thomas :

Pour cette première soutenance , Thomas était en charge de réalisé un solver de Sudoku et de le rendre capable de lire un certain type de format de fichier et de renvoyer ce sudoku résolu au même format.

Il a aussi réalisé le réseaux de neurones afin de pouvoir présenté une preuve de concept du XOR au jury.

Guy Noé :

Ma tache pour cette de première soutenance était de detecter la grille de sudoku et d'enregistrer chaque case sous forme d'image

Emre :

Ma tâche concernant cette première soutenance était de faire les premières étapes des pré-traitements de l'image : les niveaux de gris, le floutage et la binarisation de l'image. Cette étape est importante pour la détection de la grille sur l'image.

2.3 Progression du projet

Pour cette première soutenance :

Le chargement d'une image, suppression des couleurs, rotation manuelle de l'image, découpage de l'image, implémentation de l'algorithme de résolution d'un sudoku (solver), réseau de neurones (XOR) on était effectué.

Pour la détection de la grille et de la position des cases qui était une des parties les plus complexes, nous somme parti sur une idée qui n'a pas abouti c'est pourquoi pour la prochaine soutenance nous avons pensé à une tout autre méthode plus simple à implémenter (avec la transformer de Hough.

3 Pré-traitement de l'image

3.1 Pourquoi faisons-nous un pré-traitement de l'image ?

Pour une détection optimale de la grille, les pré-traitements de l'image sont nécessaire. Étant donné que les pré-traitements sont une chose nouvelle pour nous : nous devons nous renseigner et lire sur les étapes d'un pré-traitement réussie. Nous commencerons alors en suivant ces étapes-ci :

- Transformer en niveau de gris
- Appliquer un filtre de floutage
- Appliquer en chaque pixel, un principe de threshold locale
- Appliquer une rotation si nécessaire

3.1.1 Le niveau de gris

Avant de continuer dans les pré-traitements de l'image, nous devons transformer l'image en niveau de gris. Nous utilisons donc un simple programme qui permet, à chaque pixel, en sortir un pixel en niveau de gris. Cela a été permis avec cette formule, pour les couleurs RGB donné par un pixel alors ce même pixel en niveau de gris sera redéfinis en (R+G+B) / 3 pour ses 3 couleurs.

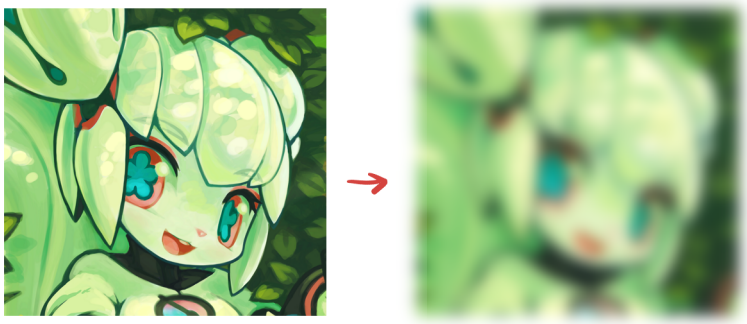
Voici ci-dessous l'image d'un sudoku transformé en niveau de gris.

3.1.2 Le Floutage

Première étape : Nous devons flouter l'image. Il y a beaucoup de techniques de floutage. Nous avons tester premièrement le floutage de Gauss. Le floutage de Gauss est très efficace. Le floutage est assez importante avant une binarisation : cela permet de réduire le niveau de bruit efficacement.
Le floutage de Gauss fonctionne avec un premier Kernel qui est identique à chaque pixel. Ce kernel est une sorte de moyenne entre le pixel et ses pixels aux alentours. Vous trouverez ci-dessous une image représentant le floutage de Gauss avec un niveau de profondeur de 15pixels.

Pour une exécution des programmes avec une rapidité optimale, le floutage de Gauss est assez complexe mais assez simple à mettre en place.

C'est pour cela que nous utilisons, à la place du floutage de Gauss, le floutage médian qui fonctionne avec similitude au floutage de Gauss à une différence : le kernel utilisé dans le floutage médian a tous ses coefficients égalent à 1. Cela permettra d'utiliser l'image intégrale (que nous verrons plus tard) qui facilitera grandement la complexité du programme et son temps d'exécution.



3.1.3 L'image intégrale

La seconde étape après le floutage est de construire une image intégrale de l'image. Le principe est assez simple, pour chaque pixel aux coordonnées XY, nous additionnons ses pixels voisins (X-1, Y et X, Y-1) moins le pixel aux coordonnées X-1 Y-1. Comme nous le montre l'image ci-dessous. L'image intégrale est très importante pour des pré-traitements efficaces : cela nous réduit considérablement le temps de calculs et la complexité de l'image.
Pour la binarisation, nous ferons aussi une image intégrale au carré, cela permettra de calculé en chaque pixel la variance de la moyenne des pixels aux alentours.

1	1	1
1	1	1
1	1	1

Image d'entrée

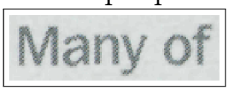
1	2	3
2	4	6
3	6	9

Image intégrale

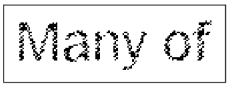
3.1.4 La binarisation de l'image

Après avoir transformé l'image en niveaux de gris et après avoir appliquer un filtre de floutage à l'image, nous appliquons une méthode de binarisation pour transformer l'image en noir et blanc pur. Après avoir étudier beaucoup d'étude et de revue scientifique, nous en avons conclus que la technique de la binarisation de Sauvola est l'une des meilleurs techniques au sujet d'un OCR. Le principe est très simple, nous utilisons un niveau de seuil unique en chaque pixel. Ce niveau de seuil est calculé à l'aide d'une formule où Sauvola y mélangent moyenne et l'écart-type des pixels environnants le pixel X.
C'est pour cela que nous avons calculé précédemment les images intégrales. Si nous devions calculé 60x60 pixels en chaque pixel, le temps de calculs sera incroyablement élevé alors qu'avec les images intégrales le temps de calcul est complètement presque

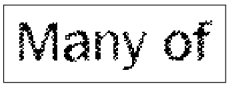
instantané en chaque pixel.



(a) Original grayscale image with textured text.



(b) Sauvola's binarization applied on (a) with $w = 11$.



(c) Sauvola's binarization applied on (a) with $w = 51$.

(d) Original grayscale image with small text.

(e) Sauvola's binarization applied on (d) with $w = 11$.

(f) Sauvola's binarization applied on (d) with $w = 51$.

(g) Original grayscale image with large text.

(h) Sauvola's binarization applied on (g) with $w = 51$.

(i) Sauvola's binarization applied on (g) with $w = 501$.

3.2 Rotation de l'image

Dans le pré-traitement nous avions également une rotation d'image à effectuer. Il s'agit d'une partie, elle aussi, importante puisqu'elle requiert d'être capable de corriger une image de travers. Ceci en prévision de l'éventualité où , l'image aurait été mal scannée, ce qui peut entraîner une détection de la grille très compliquée à réaliser, il est donc nécessaire d'avoir une bonne rotation d'image. Pour cela, on a décidé d'implémenter la librairie SDL qui nous permet de manipuler les images plus simplement.

Principe :

Pour cette première soutenance Younes a réalisé une rotation manuelle c'est-à-dire que l'utilisateur doit au préalable entrer une valeur qui correspond à la valeur de l'angle de rotation souhaité.

Et Ainsi, lorsqu'on lance la rotation, on transforme d'abord la valeur de l'angle saisie en radiant avec la formule : angle° $\times$ /180 = rad à l'aide de la macro M$_P I$.

Ensuite on calcule la taille de l'image de destination ce qui permettra par la suite d'éviter le dépassement qui a été une des difficultés lors de la réalisation de la rotation.

Par la suite on alloue la mémoire à l'espace de la surface destination. Après avoir vérifié que la mémoire a bien été allouée , on effectue le calcul du centre de l'image en divisant par deux la hauteur ainsi que la largeur de destination.

Après quoi, on détermine la valeur du pixel qui correspond le mieux pour la position i,j de la surface de destination.

Et enfin, on détermine la meilleure position sur la surface d'origine en appliquant une matrice de rotation inverse à l'aide des fonctions get-pixel et put-pixel vu dans les Tp qui permettent respectivement de déterminer la valeur d'un pixel aux positions x,y et d'écrire un pixel aux positions x,y.

Mise en pratique visuelle :

Après que l'utilisateur a saisi un angle, on charge l'image en format .jpg, puis on l'affiche.

— Affiche l'image

— Attends qu'une touche soit appuyée

— Effectue la rotation de l'image

— Affiche l'image après rotation

— Effectue une mise à jour des surfaces

— Attends qu'une touche soit appuyée

— Libère la surface de l'image

— Libère la surface de la fenêtre

4 Détection de la grille et découpage de l'image

L'idée de la détection de la grille est de pouvoir détecter la grille dans une image de sudoku afin pouvoir identifier chaque case du jeu.
La détection et la sauvegarde de la grille se déroulent alors en trois étapes principales qui sont la détection du plus grand carré puis des carres internes et enfin la sauvegarde de chaque petit carré

Pour effectuer la détection il faut que le prétraitement ait été effectué. C'est-à-dire que la binarisation et la rotation si besoin de l´image ait été faite.

4.1 Pratique :

Lorsque j´ai commencé à travailler sur cette tâche, j'ai tout d'abord regardé les méthodes généralement employées pour détecter des formes et les premiers résultats m'ont conduit vers la transformée de Hough. Mais cette méthode a été très complexe à comprendre et à mettre en application. Alors j'ai décidé de créer un algorithme qui me permettrait de remplir cette tache de plus ou moins simplement.
Le principe est le suivant (On suppose l'image déjà prétraitée.) :

— On créer une matrice de même taille que l´image.

— Pour chaque pixel de l´image on vérifie si ce pixel est blanc (resp noir)

— en vérifiant que ses valeurs RGB soit supérieur (resp inférieur) a une valeur de seuil, ici 125.

— On rempli la matrice avec des 0 et des 1 en fonction que le pixel soit blanc ou noir

— On parcourt les lignes de la matrice en recherchant les lignes de pixel.

— On parcourt les colonnes de la matrice en recherchant les colonnes de pixel. (Une ligne/colonne de pixel est définie comme une suite consécutive de 1 dans la matrice)

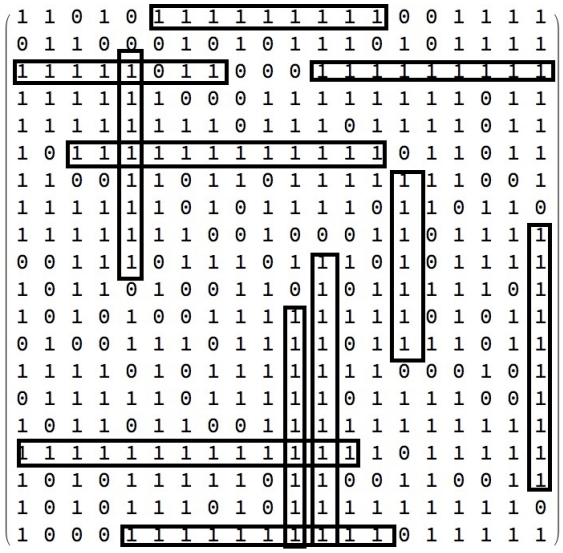
— Si la lignes/colonnes est suffisamment grande (> 10 pixel), on ajoute les coordonnées de chaque point la constituent dans un tableau des coordonnées *.

— On recherche ensuite dans le tableau des coordonnées, les coordonnes croisées, c'est a dire celle qui appartiennent à la fois au lignes et au colonnes

— On retourne ensuite les valeurs max et min de (x, y) des coordonnées croisées

— On extrait le sudoku de l'image initiale grâce à ses coordonnées

le schéma ci dessous est une représentation des lignes et colonnes et des coordonnées croisées. saut que dans notre cas, les lignes et colonnes ne sont pas aussi dispersés.



Représentation des lignes et colonnes et des coordonnées croisées.

4.2 Difficultés

L'un des premiers problèmes qu'a rencontré cet algorithme est la taille du tableau de coordonnée*. En effet le nombre de ligne et colonne que l'on peut trouver dans une image n'étant pas connu a l´avance, et étant donnée qu'une fois le tableau déclarer on ne peut ni l'agrandir ni le rétrécir, il a fallu trouver une alternative. L'alternative a été d'utiliser les listes, qui contrairement aux tableaux peuvent être extensibles. Mais le type liste n'étant pas définit par défaut, j'ai donc commencé par l'implémenter.

J'ai ainsi créé la structure liste et des méthodes de bases qui lui sont associé (déclaration, ajout et suppression d'élément dans la liste...) et enfin finaliser l'algorithme de détection de la grille.

4.3 Résultats :

Actuellement le résultat obtenu n'est pas concluant. En effet, pour une raison que je n'arrive pas à expliquer, le programme ne parcoure pas l'entièreté de la matrice. L'on parvient à extraire une image qui n'est pas celle attendu. De plus l'exécution du programme est un peu lente (14 secondes environ) dû sa complexité et crée des leaks mémoires qui eux aussi restent inexpliqué.

4.4 Améliorations :

Une amélioration qui va être faite est de changer de méthode de détection, en me tournant vers permet de détecter les lignes présentes dans une image. En caractérisant les droites par leur pente et leur ordonnée à l'origine. la méthode de flood fill segmentation qui est algorithme permettant d'identifier et/ou de modifier les valeurs adjacentes dans une image en fonction de leur similitude avec un point de départ initial. Pour avoir une image plus parlante, celà correspond à l'outil seau de remplissage dans Paint.

5 Implémentation d'un réseau de neurone

5.1 Recherche

Pour commencer , Thomas ne savait absolument pas par quoi commencer pour réaliser un XOR , en effet , la seule expérience avec les réseaux de neurones qu'il avait , était le flappy bird réalisé en Csharp a la fin du Semestre 2. TP qu'il a absolument raté tellement le concept de réseau de neurones était flou pour lui.

Pour cela , il a réalisé ce que tout apprenti programmeur réalise , il a cherché sur internet des exemples de réseaux de neurones sur internet. Cet opération fut peu fructueuse , en effet les connaissances de Thomas sur les réseaux de neurones après cette opération ont augmenté de 0 pourcent , il fallait donc reprendre tout de 0 et comprendre

comment marchait un réseaux de neurones en général, et non seulement pour le Xor .

Pour cela des personnes de la classe ont conseillé à Thomas de prendre finalement une approche Mathématique de la chose. Pour cela un site internet donné dans les sources a permis à Thomas de comprendre le réseau de neurones dans son entièreté

Principe de fonctionnement :

- Le Xor utilise l'algorithme de Forward propagation afin de calculer la valeur des Output ainsi que le taux d'erreur.
- Pour mettre à jour les poids , l'algorithme réalise une backpropagation
- Tout d'abord il fallait calculer la valeur des noeuds caché en faisant l'opération :
  Pour i allant de 0 à Nombre d'input
  Somme de (Poids de la liaison Input i -> hidden multiplié par la valeur de l'input en question) additionné au biais entre input et Hidden
- Pour calculer la valeur de l'output du hidden layer i , on récupère la somme précédente et on y applique la fonction de notre choix , ici Thomas à choisi d'utiliser la fonction sigmoid car c'est ce que tout le monde conseillait d'utiliser.
- On répète ce processus en remplaçant les inputs par les output de la couche cachée et on obtient un résultat final.
- Ce résultat , on vas l'utiliser pour calculer le taux d'erreur avec la fonction $\frac{1}{2}$(cible - resultat) le tout au carré on réalise la backpropagation pour corriger les valeurs des poids , les opérations étant d'une longueur excessive. Elles ne seront pas explicité dans ce rapport

5.2 Difficulté et Amélioration :

Lors de la définition des arrays , Thomas à tenté de réaliser une allocation dynamique de la mémoire mais cela lui à donné des Segfault et des memory leaks , il s'est donc tourné vers une implémentation statique. Pour la prochaine soutenance , le but pour lui sera de réalisé une application dynamique lui permettant de créer des réseaux de neurones de tailles variables.

6 Résolution du Sudoku

6.1 L'algorithme

Pour la réalisation du solver l'implémentation de l'algorithme était libre , c'est-à-dire que Thomas pouvait réaliser n'importe quel algorithme , que cela soit un algorithme de résolution par Réseaux de neurones ou alors de manière récursive.

Après avoir réfléchi , Thomas à décidé d'utiliser la méthode la plus simple , celle que les élèves ont déjà créé pendant les TP du S2 ,la manière récursive.

Principe :

Le principe de l'algorithme est très simple , le programme prend un nombre entre 1 et 9 et le pose à la case libre où se trouve l'index de l'algorithme et réalise une série de test sur la ligne , la colonne , ainsi que sur la matrice 9x9 lié à cet index.

Si tout ces tests renvoient vrai , alors dans ce cas le nombre est validé et le programme continue récursivement sur une autre case. Cet algorithme permet de gérer les cas où l'algo a mal placé son premier nombre , car trop peu d'information au préalable, notamment grace à la remontée de la récursive.

Après avoir implémenté l'algorithme en traduisant les composantes de CSharp en C. Le programme ne fonctionnait absolument pas , il y avait bien des parcours de matrice qui se réalisait mais cela renvoyait les mauvaises valeurs , l'erreur fut trouvé plus tard, le système de matrice de type grid[SIZE][SIZE] ne prend pas se arguments de manière grid[x][y] mais plutôt de manière grid [y][x]. Après avoir inversé les variables, le programme fonctionnait correctement.

6.2 Le reader

Lors de la conception du reader , Thomas pensait que cela allait être une tâche assez facile étant donné que la création de filestream pour lire les données d'un document ont été une partie importante des tp l'année passé,c'était une grossière erreur.

Principe de fonctionnement :

Le programme prend en argument le chemin d'accès du fichier ainsi que la matrice qui doit être complété. Il l'ouvre et lit chaque caractère 1 par 1 tout en incrémentant 2 variables permettant de se déplacer dans la matrice . Si le caractère n'est pas un chiffre ou un point , alors il est ignoré , si le caractère est un point, alors dans la matrice apparaîtra un zéro à son index. Thomas à choisi cet implémentation plutôt que de faire une matrice de caractères afin à ne pas avoir a géré des conversions de type char vers des types int lors de la résolution

Penser que cet algorithme était facile fut une énorme erreur,en c , les arguments d'une fonction sont renvoyés avec le pointeur argv. Il n'y a donc pas de string , ce qui a rendu la tâche de lire l'input donné par l'utilisateur compliqué lorsque l'on ne sait pas que argv[0] renvoie la fonction écrite dans le terminal et qu'il faut choisir argv[1] afin de recevoir le premier argument.

Ensuite, lors de la lecture des caractères , Thomas souhaitait ne pas prendre en compte le caractère "/n" ainsi que les espaces , or il semblerait que pour son cas simplement les exclures ne fonctionnait pas et qu'il fallait vérifier chaque lettre avec le code ASCII de ce que l'on souhaite ne pas prendre en compte dans l'algorithme.

De plus, le problème d'agencement des matrices n'était pas connu par Thomas lors de l'implémentation du code , donc il recevait des grilles fausses , à l'envers et a réussi à réparer ça après la découverte.

6.3 Le writer

Le programme permettant d'écrire dans un fichier le résultat de la matrice dans un fichier était de loin le plus facile , notamment car Thomas l'a implémenté à la toute fin , lorsque tous les autres programmes liés au solver étaient fonctionnels. Et donc qu'il avait compris comment marchait les filestream et les matrices en c. Son principe est très simple , il lit la matrice grid et réalise une simple fonction print dans le fichier , en ajoutant des caractères "/n" et des espaces avec des conditions

6.4 Difficulté

Lors de la première implémentation de cet algorithme , Thomas est parti sur un des sous fonctions qui renvoie directement la grid , avec comme prototype de fonction int *reader par exemple , cette implémentation prenait énormément de mémoire car chaque matrice était recrée, donnant donc à la fin de l'execution du programme 5 matrice de taille 9 fois 9 dans la mémoire. Thomas a donc été conseillé par certaines personnes extrêmement compétente dans la classe , d'utiliser des types void et de faire 1 seule matrice. Ce qui permet aujourd'hui d'avoir un programme qui crée 1 seule matrice et qui la modifie.

7 Conclusion

À l'issue de cette soutenance intermédiaire, l'essentiel de l'OCR a été réalisé. En effet,nous avons réalisé un réseau de neurones capable d'apprendre la fonction logique XOR, il est d'ailleurs plus avancé que cela pour être applicable directement dans un OCR.Bien que ce projet ne soit pas aussi abouti que l'on l'espérait, il nous a été bénéfique en de nombreux points tels que la découverte et l'approfondissement nouvelles notions (réseaux de neurones, binarisation de l'image et les listes chaînées), la participation à un tel projet de groupe d'envergure, la découverte de nouvelle approche de travail. Ce projet nous à tous appris à faire un vrai travail d'un ingénieur : celui de rechercher les bonnes études concernant un problème donné.

8 Références et Sources

— 1. M. Sezgin and B. Sankur. Survey over image thresholding techniques and quantitative performance evaluation. Journal of Electronic Imaging, 13 :146–165, 2004.

— 2. N. Otsu. A threshold selection method from gray-level histograms. IEEE Transactions on Systems, Man, and Cybernetics, 9(1) :62–66, January 1979.

— 3. J. Bernsen. Dynamic thresholding of grey-level images. In Proceedings of the International Conference on Pattern Recognition, pages 1251–1255, 1986.

— 4. E. Gabarra and A. Tabbone. Combining global and local threshold to binarize document of images. In Pattern Recognition and Image Analysis, volume 3523 of LNCS, pages 173–186. Springer, 2005.

— 5. Y. Rangoni, F. Shafait, and T. M. Breuel. OCR based thresholding. In Proceedings of IAPR Conference on Machine Vision Applications, pages 98–101, 2009.

— 6. https ://en.wikipedia.org/wiki/Hough$_{t}ransform$.
7.$https$: $//en.wikipedia.org/wiki/Flood_{f}ill$.

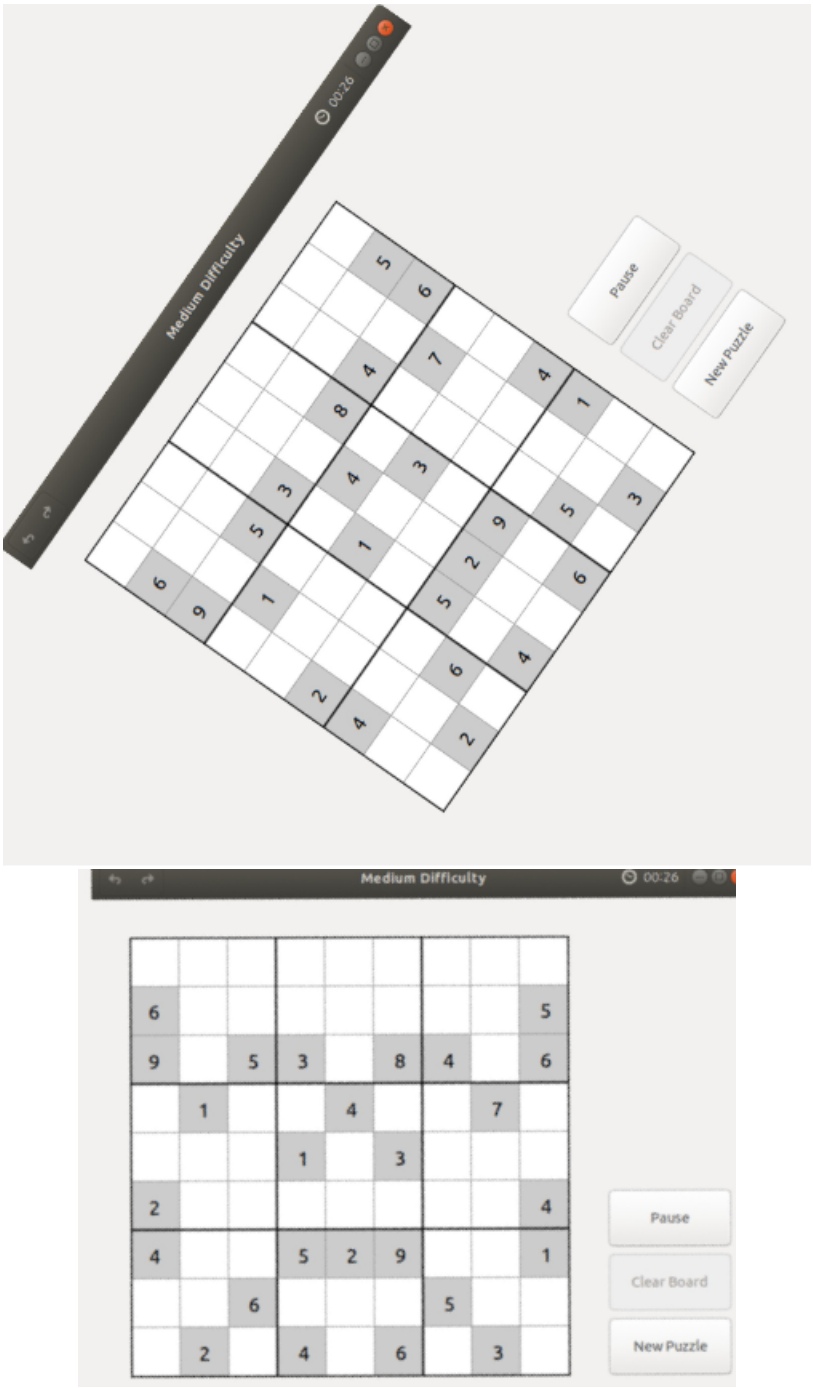


FIGURE 1 – Exemple de roation d'image.