UQAC

Université du Québec
à Chicoutimi

8INFF804 : Traitement numérique des images

Hiver 2022

Projet – Solveur de sudoku par lecture d'image

Paul DELARUE - DELP02049907
Julia MOURIER - MOUJ21579909
Martin MORAND - MORM16089908

Le lien GitHub pour ce TP : https://github.com/JuliaMourier/Traitement-d-image-Projet

Semestre Hiver 2022

Table des matières

I. Exécuter le programme

Le programme est écrit en python 3.9. Il lui faut donc cette version de python pour être lancé.

Nous utilisons la bibliothèque numpy (https://numpy.org/install/), ainsi que openCV (https://sourceforge.net/projects/opencvlibrary/), tkinter (https://www.tutorialspoint.com/how-to-install-tkinter-in-python) et également argparse (https://pypi.org/project/argparse/).

De plus, les bibliothèques tensorflow et keras sont nécessaires pour lancer notre programme. Pour les installer vous pouvez lancer la commande « pip install tensorflow » avec pip. Si ces commandes ne sont pas suffisantes, vous pouvez vous renseigner sur le site : https://www.activestate.com/resources/quick-reads/how-to-install-keras-and-tensorflow/

Nous avons développé le code source du TP avec l'IDE PyCharm. Pour le lancer sur PyCharm il suffit de presser le bouton run en haut de l'IDE avec les bons arguments. Vous pouvez également lancer la commande dans un terminal.

Le fichier à lancer est le suivant : « main.py ». Vous devez ajouter le chemin du l'image du sudoku à résoudre en paramètre soit dans l'IDE ou bien en ligne de commande : « python main2.py <path_of_the_image> »

L'image dont le path est donné doit pour le mieux contenir seulement la grille du sudoku avec seulement une petite marge autour de la grille afin de garantir une bonne récupération de la grille. Une bonne luminosité sur l'image conduit à de meilleurs résultats.

A son lancement, l'algorithme va commencer par afficher l'image entrée, puis après l'appuie sur une touche du clavier, il va afficher la grille récupérée. Après un temps d'attente, le sudoku s'affiche en console sous la forme d'un tableau de taille 9x9 et le programme demande à l'utilisateur s'il est satisfait des résultats de la récupération. Une fois que l'utilisateur a vérifié que le sudoku est bon, l'algorithme de résolution se lance et une fenêtre contenant le sudoku résolu s'affiche si ce dernier peut être résolu. S'il ne peut pas être résolu, l'algorithme l'indique dans la console.

II. Notre choix de sujet

Nous avons choisi de partir sur un solveur de sudoku par lecture d'image car nous trouvions amusant de pouvoir résoudre un sudoku en scannant une photo. Nous avions également la partie résolution que nous avions réalisé dans le cours d'IA et nous avions envie de porter ce projet plus loin.

Ainsi, notre projet pourrait permettre de vérifier une solution ou de résoudre le sudoku.

Nous avions pensé dans un premier temps entrer une grille de sudoku, récupérer chaque chiffre puis entrer le sudoku dans un algorithme de résolution qui se base sur un algorithme de backtracking avec satisfaction de contrainte (CSP), pour finir par afficher le sudoku.

Nous avons été plus loin dans notre pensée en ajoutant la possibilité de rentrer une image qui ne contient pas seulement la grille mais aussi une marge autour de la grille.

L'idée principale est de faire la reconnaissance des chiffres par un réseau de neurone après avoir segmenté chaque case du sudoku.

III. Choix d'implémentation

La librairie pour la partie traitement d'image et préparation d'image est openCV. Ainsi, les modules de segmentation d'image, de récupération de la grille et de traitement sont faits avec openCV.

La partie reconnaissance de chiffre est traitée par un réseau de neurone convolutif avec tensorflow keras.

IV. Présentation du dataset

Nous avons dû utiliser un dataset pour faire apprendre le réseau de neurone. Comme le but du réseau de neurone est de reconnaitre des chiffres et de les classifier, nous avons utilisé le dataset mnist fourni avec keras. Les chiffres de ce dataset sont sous forme d'images de taille 28 par 28. Mais comme nos données ne sont pas de cette taille-là, nous avons prétraiter le dataset pour que les données correspondent à celles attendues.

```
    # load dataset
    (X_train, y_train), (X_valid, y_valid) = mnist.load_data()

    new_X_train = []
    new_X_valid = []

    for train in X_train :
        #new_X_train.append(prepareImage(train))
        new_X_train.append(cv2.resize(train, (50,50), interpolation=cv2.INTER_AREA))

    for valid in X_valid :
        #new_X_valid.append(prepareImage(valid))
        new_X_valid.append(cv2.resize(valid, (50,50), interpolation=cv2.INTER_AREA))

    new_X_train = np.array(new_X_train)
    new_X_valid = np.array(new_X_valid)

    # reshape input to one channel
    new_X_train = new_X_train.reshape((new_X_train.shape[0], 50, 50, 1))
    new_X_valid = new_X_valid.reshape((new_X_valid.shape[0], 50, 50, 1))

    # normalize values
    new_X_train = new_X_train / 255
    new_X_valid = new_X_valid / 255

    # one hot encode target values
    y_train = to_categorical(y_train)
    y_valid = to_categorical(y_valid)
```

La capture d'écran ci-dessus montre la partie du code où nous récupérons le dataset. Ensuite nous changeons la taille des images. Le réseau de neurone a été entrainé plusieurs fois sur le dataset mais celui-ci a été traité de deux manières. Tout d'abord nous n'avons changé que la taille des images et avons sauvegardé les poids du réseau de neurone. Puis le dataset a été traité par la fonction "prepareImage" qui change la taille et qui effectue des opérations sur chaque image. Dans ces opérations on peut compter un flou gaussien et de l'érosion. Cette fonction nous sert à traiter les segments d'images que nous récupérons, nous avons trouvé judicieux de passer les données d'entrainement dans cette fonction afin que ces dernières concordent le plus possible avec les données réèllement utilisées.

Comme le but du réseau est de catégoriser un chiffre, nous utilisons la fonction "toCategorical" sur les données de sorties de l'entrainement afin d'avoir nos sorties en one hot encoded.

V. Fonctionnement de l'algorithme

A. *Traitement de l'image d'entrée*

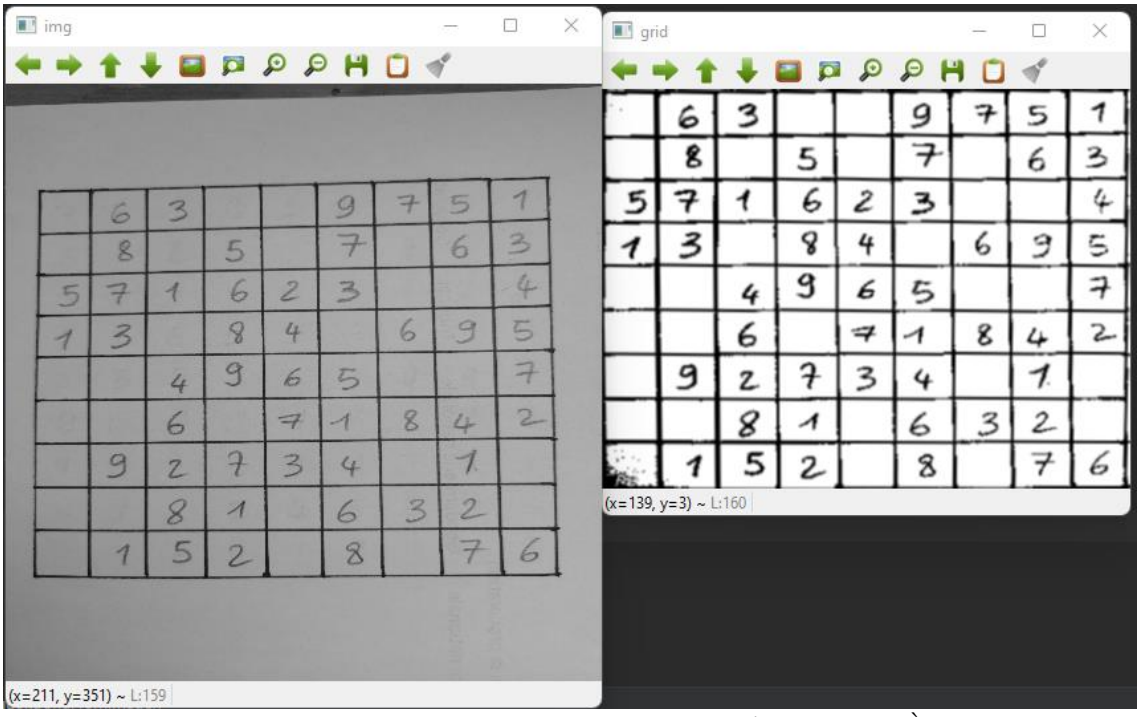
Après avoir récupéré le chemin de l'image en argument du programme grâce à la librairie argparse, nous avons choisi de réalisé un premier traitement sur l'image.

L'algorithme commence par changer la taille de l'image à 450x450 pixel, puis affiche l'image.

La fonction getSudokuGridFromImage() permet dans un premier temps de récupérer seulement la grille de sudoku et de cadrer l'image dessus. Pour cela, la première étape est un passage en nuance de gris si elle ne l'est pas déjà. Puis, on effectue un flou gaussien. Cet effet de flou suivit d'un seuil adaptatif va permettre de contraster la grille afin de récupérer son contour. C'est ce que l'on fait ensuite grâce à la fonction findContours(). En fait, on récupère tous les contours de l'image puis on ne prend que le plus grand contour trouvé. Ce dernier contour est le tour de la grille.

```
def getSudokuGridFromImage(image: np.ndarray):
    """
    function that takes an image, finds the sudoku grid and returns the area found
    """

    # copy image
    img = image.copy()

    # check if sudoku is in grayscale, if not : transform to grayscale
    if not len(img.shape) == 2:
        img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # apply gaussian blur
    blur = cv2.GaussianBlur(img, (5, 5), 0)
    # apply gaussian threshold
    thresh = cv2.adaptiveThreshold(blur, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
                                   cv2.THRESH_BINARY_INV, 11, 2)
    # find all the contours
    contours = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
                                cv2.CHAIN_APPROX_SIMPLE)[0]
```

Une fois le contours de la grille récupéré, on approxime un polygone correspondant au tour de la grille. Le but ensuite est de remettre la grille dans la bonne perspective. Pour cela, on calcule le cadre de sélection approximé précedemment ainsi que le rectangle de destination pour utiliser les fonction getPerspectiveTransform() et wrapPerspective(). La fonction getSudokuGridFromImage() se termine enfin par un gaussian blur afin de remplir les potentiels trous dans les chiffres et dans la grille.



L'image ci-dessus montre un exemple de récupération d'une grille. À gauche on trouve l'image donnée initialement à l'algorithme, à droite on trouve la grille récupérée.

B. Segmentation

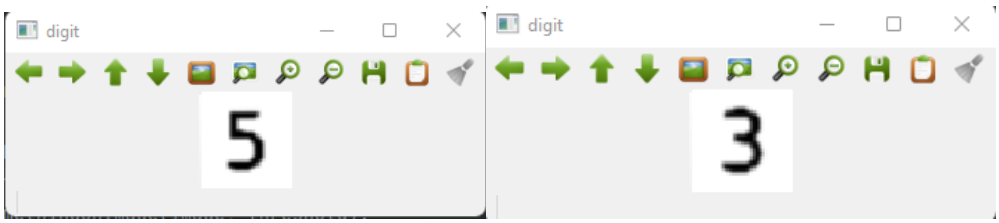
L'étape suivante est la segmentation. L'idée est de séparer la grille de sudoku en 81 carrés pour chaque numéro de la grille. C'est la fonction **getSegmentedSudoku**() qui va permettre de réaliser ceci.

```
92  def getSegmentedSudoku(img: np.ndarray):
93      # get contours
94      binary = cv2.threshold(img, 225, 255, cv2.THRESH_BINARY)[1]
95      ext_contours, hierarchie = cv2.findContours(binary, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
96      segmentedSudoku = []
```

On effectue d'abord un threshold afin de ne travailler qu'avec des images en noir et banc. Puis, on utilise une nouvelle fois la fonction **findContours**() qui cette fois permet de récupérer tous les contours du sudoku. Ces contours sont les contours dit "extérieurs", ils correspondent aux cases du sudoku.

```python
for contour in ext_contours:

    # find corners
    max_x = contour[0][0][0]
    min_x = contour[0][0][0]
    max_y = contour[0][0][1]
    min_y = contour[0][0][1]
    for point in contour:
        if point[0][0] > max_x:
            max_x = point[0][0]
        if point[0][0] < min_x:
            min_x = point[0][0]
        if point[0][1] > max_y:
            max_y = point[0][1]
        if point[0][1] < min_y:
            min_y = point[0][1]

    if max_x - min_x < 20 or max_y - min_y < 20:
        continue

    pt1 = [min_x, min_y]
    pt2 = [min_x, max_y]
    pt3 = [max_x, max_y]
    pt4 = [max_x, min_y]

    new_size_x = max_x - min_x
    new_size_y = max_y - min_y

    inputs_pts = np.float32([pt1, pt2, pt3, pt4])
    outputs_pts = np.float32([[0, 0], [0, new_size_y], [new_size_x, new_size_y], [new_size_x, 0]])

    M = cv2.getPerspectiveTransform(inputs_pts, outputs_pts)
    segment = cv2.warpPerspective(img, M, (new_size_x, new_size_y))
```

Pour chaque contours récupéré, nous retrouvons leur quatres coins afin d'extraire les case de la grille. Nous les extrayons une nouvelle fois grâce à la fonction wrapPerspective().



Les deux images ci-dessus sont des exemple de cases récupérées. Ces cases ne sont pas affichées par l'algorithme.

```python
        # find center position on sudoku grid
        moment = cv2.moments(contour)
        # center [x,y]
        if moment["m00"] == 0 :
            continue
        centerx = int(moment["m10"] / moment["m00"])
        centery = int(moment["m01"] / moment["m00"])

        segmentedSudoku.append((segment, centerx, centery))

    sortedSegmentedSudoku = sortFoundNumbers(segmentedSudoku)
```

Comme les contours ne sont pas forcément ordonnés dans l'images, il arrive que l'algorithme les ordonne de manière aléatoire. Nous avons donc décidé de créer une méthode permettant de trier les images récupérées en fonction des positions de leur centre. Nous commençons par calculer le centre de chaque case segmentée puis le sauvegardons dans une liste. Cette liste est ensuite passée dans la fonction suivante :

```
158  def sortFoundNumbers(list):
159      sortedList = list.copy()
160      # sort by y
161      sortedList = sorted(sortedList, key=lambda y: y[2])
162      # sort by x
163      for i in range(0,9) :
164          sortedList[i*9:(i+1)*9] = sorted(sortedList[i*9:(i+1)*9], key=lambda y: y[1])
165      return sortedList
```

Cette méthode tri d'abord les éléments selon leur position en y afin de remettre chaque case dans la bonne rangée, puis chaque rangée recréée est triée suivant les positions en x. On obtient un tableau contenant les images triées de gauche à droite puise de bas en haut.

On vient ensuite placer dans une liste seulement les images des cases segmentées. On obtient ainsi le sudoku segmenté.

C. *Lecture des digits*

Cette étape nécessite un réseau de neurone pour distinguer les numéros dans les cases. La fonction que nous avons créée pour remplir ce rôle est MakeSudokuMatrice().

```
7   def MakeSudokuMatrice(listOfImages: [np.ndarray]):
8       """
9       take a list of image, find the value in it and returns a sudoku as a matrice 9x9
10      :param listOfImages: should be a list of np.ndarray of size [9][9]
11      :return:
12      """
13      listOfValues = []
14
15      threshold = 240
16
17      for image in listOfImages:
18          value = 0
19          cv2.imshow("digit", image)
20          cv2.waitKey(0)
21          if isSomethingOnImage(image, threshold):
22              value = findDigit(image)
23          #print(value)
24          listOfValues.append(value)
25
26
27      sudoku = turnListIntoSudoku(listOfValues)
28      return sudoku
```

Afin de trier les cases vides des cases contenants des numéros, nous vérifions si la moyenne de valeur de gris des pixels de la case est supérieure à un seuil (ici choisi empiriquement à 240). S'il y a bien un numéro, le réseau de neurone est appelé via findDigit() afin de déterminer le numéro associé. S'il n'y a pas de numéro on associe la valeur 0 à la case qui sera compris comme une case vide par le solveur.

```python
def findDigit(image: np.ndarray):
    """
    function that takes an image and returns un number between 0 and 9 :
    number on image should be written black on white background

    - get pretrained model
    - prepare image to fit in NN input
    - get prediction from model
    - find associated number from one hot encoded prediction
    - return value

    function is to use on every part of sudoku found in processing scripts
    """

    # get model
    # MODEL SHOULD BE TRAINED - model is trained if "digitModel" directory exists
    model = digitFinderModel(False)
    # resize image to predict
    img = prepareImage(image)

    prediction = model.model.predict(img)

    number = getPredictionValue(prediction[0])

    return number
```

Le réseau de neurones utilisé est un réseau de neurones convolutif que nous avons créé et entrainé sur le dataset mnist comme expliqué dans la partie IV de ce rapport. Le réseau de neurone a l'architecture suivante :

```python
class digitFinderModel:
    """
    CNN trained on mnist dataset to categorise numbers

    numbers are written in white with black background of size (50,50,1)

    don't forget to normalize the input image

    CNN is :
        - conv2D(32,(3,3), relu)
        - MaxPooling2D(2,2)
        - Flatten
        - Dense(100, relu)
        - Dense(10, softmax)

    out is one hot encoded
    """

    def __init__(self, toTrain: bool):
        self.model = Sequential()

        self.model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(50, 50, 1)))
        self.model.add(MaxPooling2D((2, 2)))
        self.model.add(Flatten())
        self.model.add(Dense(100, activation='relu'))
        self.model.add(Dense(10, activation='softmax'))

        if (toTrain):
            self.trainModel()
```

Nous avons pensé que ces quelques couches étaient suffisantes pour les besoins de ce projet. En réduisant le nombre de couche, nous avons pensé que l'entrainement serait plus facile et que la prédiction de chiffre prendrait moins de temps.

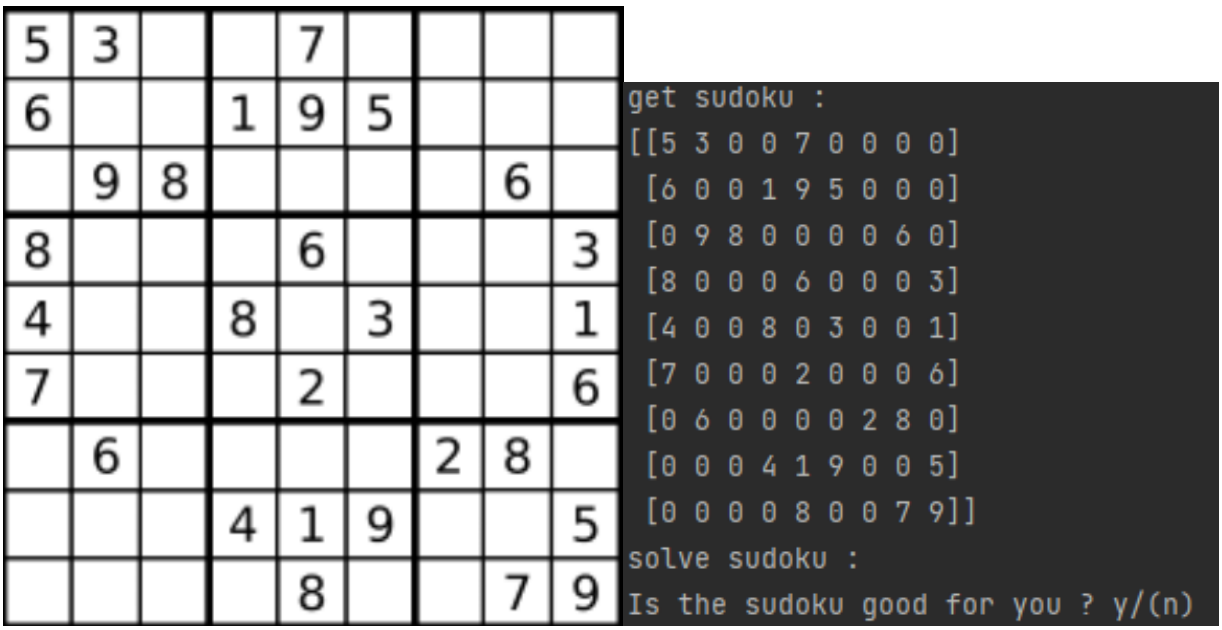
Chaque valeur est ajoutée dans une liste qui sera ensuite reconstituée en sudoku par la fonction turnListIntoSudoku(). Nous récupérons un tableau numpy de taille 9x9.

On obtient alors le sudoku à résoudre.

D. Vérification des valeurs dans le sudoku

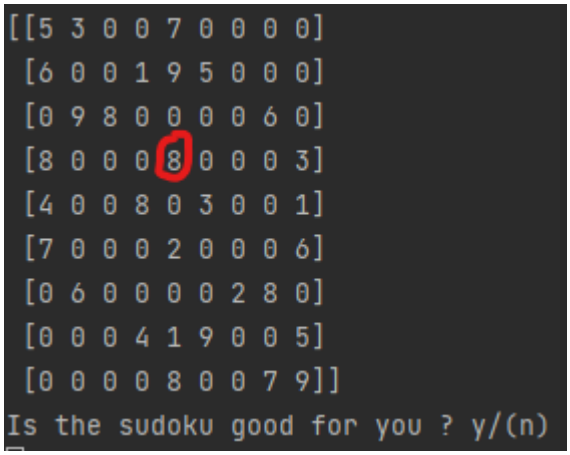
La détection des nombres pouvant commettre des erreurs, nous avons décidé de créer une fonction de redressement de solution. Une fois le sudoku récupéré, l'algorithme affiche le sudoku en console et demande à l'utilisateur si les chiffres affichés sont exacts. Si l'utilisateur n'est pas content, il peut demander à remplacer des valeurs en entrant la position de la valeur à changer. L'utilisateur peut changer autant de valeur qu'il souhaite jusqu'à être satisfait du résultat.

Par exemple, pour le sudoku suivant le programme affiche en console le résultat correspondant :

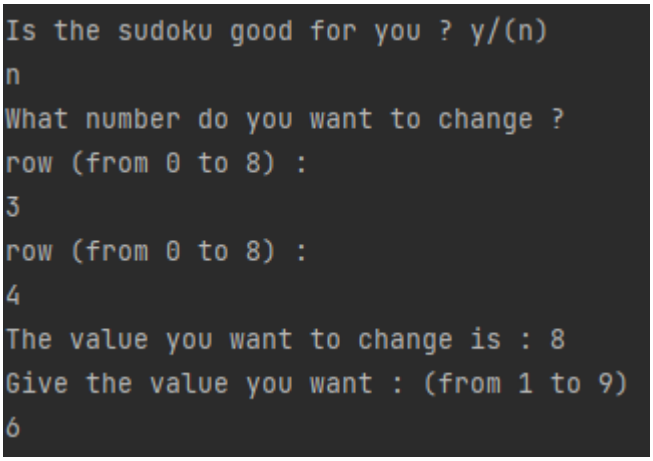


Ici le résultat est satisfaisant, l'utilisateur peut entrer y pour passer à la résolution.

Dans la capture d'écran suivante, le chiffre entouré n'est pas bon :



Si on veut le changer, on entre “n” et on donne la position 3 pour la rangée et 4 pour la colonne comme dans la capture d’écran suivante :



L’algorithme affiche ensuite le sudoku corrigé et redemande à l’utilisateur s’il veut changer une autre valeur jusqu’à ce qu’il soit satisfait.

E. Solveur de sudoku

L’étape suivante est la résolution du sudoku. Nous avons réutilisé un algorithme réalisé dans le cadre d’un TP d’IA. Celui-ci fonctionne par bactracking. La résolution de sudokus est un problème à satisfaction de contrainte (CSP). L’algorithme recherche l’ensemble des solutions possible tout en évitant des solutions qui n’aboutirait à rien.

On peut voir notre algorithme comme un arbre, de profondeur le nombre de case vide du sudoku à résoudre. À chaque branche de l’arbre on ajoute un numéro en choisissant d’abord la case la plus contrainte pour optimiser. Ainsi, dès qu’une branche n’est plus viable, c’est-à-dire que dès qu’un numéro ajouté ne correspond pas aux règles du sudoku, la branche est arrêtée. Cela permet de diminuer le nombre d’itération et de ne pas continuer sur des branches infertiles.

```python
# Main part, called by main
def backtracking_csp_algorithm(sudoku: []):
    # Display the depth of the algorithm, (9 ^ number of empty squares)
    nb_empty_squares = number_of_empty_squares(sudoku)
    print("Depth : " + str(nb_empty_squares) + " empty squares => 9^" + str(nb_empty_squares) + " = " + str(
        math.pow(9, nb_empty_squares)))
    return solve(sudoku)
```

Le corps de l'algortihme est la fonction backtracking_csp_algorithm() qui affiche la profondeur du problème (9^nb_de_case_vide, car il y 9 possibilités pour chaque case). Elle appelle ensuite le solveur ; solve() qui prend en entrée le sudoku à résoudre.

```python
def solve(sudoku: []):
    # if there is no empty square :
    if is_sudoku_filled(sudoku):
        # Return if the sudoku is soluble and the solution if exists
        return is_sudoku_valid(sudoku), sudoku
    else:  # if the sudoku is not filled
        # Get the indexes of the most constrained empty square (MRV)
        i_choice, j_choice = get_most_constrained_square(sudoku)
        # Try every possibility (between 1 and 9) for this square
        for i in range(1, len(sudoku) + 1):
            sudoku[i_choice][j_choice] = i
            # If the solution is valid, continue; else pass to next numbers (allowing to test only the possible grids)
            if is_sudoku_valid(sudoku):
                # Continue to solve the new sudoku
                is_solved, sudoku = solve(sudoku)
                if is_solved:
                    # Return that the sudoku is soluble and the solution
                    return True, sudoku
        # else (if any number fit)
        # Remove the last modification
        sudoku[i_choice][j_choice] = 0
        # Return that the sudoku is insoluble
        return False, sudoku
```

La fonction solve() est une fonction récursive. À chaque itération sur elle-même, elle vérifie que le sudoku est rempli. Dans un cas positif, la solution est renvoyée avec le test de validité.

Dans un cas négatif, nous récupérons le carré le plus contraint. Pour cela, nous appelons la fonction get_most_constrained_square() qui cacule la matrice de contrainte. Celle-ci associe une valeur à chaque case correspondant au nombre de case remplie sur sa ligne, sa colonne et son bloc.

Par exemple la grille suivante donnera la matrice :

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

8	10		13	10	10	6	6	7
12	14	12				10		
		11	16			9	9	10
14					15		15	16
14		14		15	15			
	19							
				14		10	10	
		13		11	11	7	7	8
12		12		10	10	6	6	7

La case la plus contrainte récupérée, on lui associe un chiffre et si la nouvelle grille est valide, on continue le remplissage jusqu'à soit tomber sur la solution, soit par tomber sur une invalidité et dans ce cas, un autre chiffre est testé jusqu'à obtenir à coup sur un chiffre juste.

A la fin des nombreuses itérations de solve() nous obtenons soit la certitude que le sudoku ne peut pas être complété ou bien la solution de celui-ci. Il ne reste plus qu'à l'afficher.

F. Affichage du résultat

```python
import tkinter as tk

class View:
    # Take a window and a sudoku
    def __init__(self, sudoku: []):
        self.win = tk.Tk()

        self.win.title("Sudoku Solveur")
        self.sudoku = sudoku

        # Create a grid 3x3 of grids 3x3
        for y in range(0,3):
            for x in range(0,3):
                tk.Frame(self.win, highlightthickness=3, highlightbackground="black", width=80*3, height=80*3).grid(row=y, column=x)
                frame = self.win.grid_slaves(y, x)[0]
                for j in range(0,3):
                    for i in range(0,3):
                        tk.Canvas(frame, highlightthickness=2, highlightbackground="black", width=80,height=80).grid(row=j, column=i)

        # Wait 10ms and fill the canvas with the sudoku's values
        self.win.after(10, self.fill_canvas())
        self.win.mainloop()

    # Fill the canvas with the sudoku's values
    def fill_canvas(self):
        # Creation of 3x3 frames
        for y in range(0, 3):
            for x in range(0, 3):
                frame = self.win.grid_slaves(y, x)[0]
                # Create a frame for each sgaure
                for j in range(0,3):
                    for i in range(0,3):
                        can = frame.grid_slaves(j,i)[0]
                        can.delete("all")
                        # If the square is not empty
                        if self.sudoku[y*3+j][x*3+i] != 0:
                            text = str(self.sudoku[y*3+j][x*3+i])
                            # Add the number of the square of the sudoku if it is not empty at the right place
                            can.create_text(40, 40, text=text, fill="black", font=('Helvetica 30 bold'))
```

Enfin, la dernière étape est d'afficher le sudoku proprement ; ce que nous avons réalisé avec la librairie graphique tkinter. Nous avons créé une classe View qui prend un sudoku en entrée et affiche le sudoku en le regroupant en grille de case 3x3 comme souvent nous pouvons les voir imprimés.

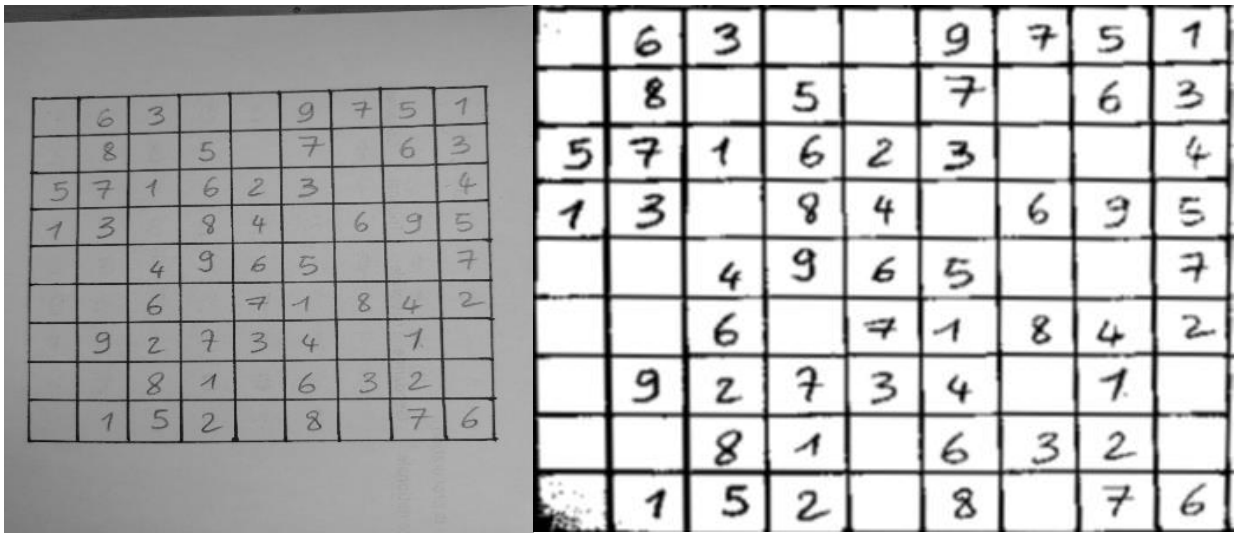
Voici un exemple de rendu :

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

VI. Résultats

Dans cette partie, nous allons réafficher les résultats pour une image donnée dans le sudoku.

Pour l'image suivante, on récupère la grille correspondante :



L'algorithme trouve le sudoku suivant après traitement :

```
[[0 6 3 0 0 9 7 5 0]
 [0 8 0 5 0 7 0 8 3]
 [5 7 1 1 2 3 0 0 0]
 [1 3 0 8 4 0 6 3 5]
 [0 0 4 9 6 5 0 0 9]
 [0 0 6 0 4 0 1 6 2]
 [0 9 8 7 3 6 0 1 0]
 [0 0 8 0 0 0 7 2 0]
 [3 7 5 2 0 8 0 9 6]]
Is the sudoku good for you ? y/(n)
```

Le résultat n'est pas très satisfaisant car certains chiffres ne sont pas bien reconnus par le réseau de neurone
Après quelques corrections, on obtient le sudoku suivant :

```
Sudoku :
[[0 6 3 0 0 9 7 5 1]
 [0 8 0 5 0 7 0 6 3]
 [5 7 1 6 2 3 0 0 4]
 [1 3 0 8 4 0 6 9 5]
 [0 0 4 9 6 5 0 0 7]
 [0 0 6 0 7 1 8 4 2]
 [0 9 2 7 3 4 0 1 0]
 [0 0 8 1 0 6 3 2 0]
 [0 1 5 2 0 8 0 7 6]]
Is the sudoku good for you ? y/(n)
```

Enfin, après avoir appuyé sur “y”, on obtient la résolution suivante :

Sudoku Solveur

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

Le sudoku est résolu.

VII. Analyse et critique des résultats

Les résultats ne sont pas toujours très satisfaisants, c'est pour cela que nous avons choisi de faire un algorithme de correction. Après analyse de notre solution, nous avons trouvé que deux points majeurs posent problème dans l'algorithme.

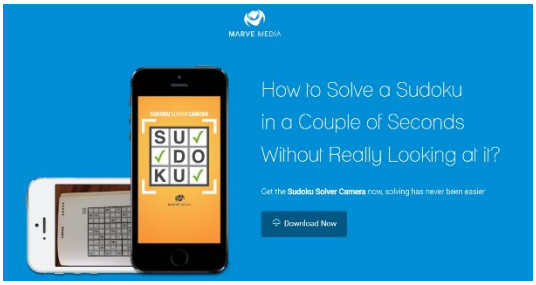
Le premier problème est la décision de l'algorithme à savoir si un chiffre est présent ou non dans la case. Nous avons choisi de faire ce choix en fonction de la moyenne de valeur de gris des pixels. Si l'image est bruitée ou que le chiffre n'est pas assez grand, il se peut que l'algorithme décide mal de la présence d'un chiffre. Dans l'exemple précédent par exemple, l'algorithme trouve un 3 en bas à gauche alors que ce qui est présent est du bruit du au passage en noir et blanc de l'image. A plusieurs endroits, il affiche 0 au lieu du chiffre voulu car le chiffre est trop petit ou écrit trop finement. Pour pallier ce problème, plusieurs possibilités existent. Nous aurions pu passer plus de temps à trouver une valeur de seuil de détection adéquat en fonction du bruit sur l'image par exemple. Ou alors nous aurions pu créer un réseau de neurone de classification binaire qui renvoie la présence ou non d'un chiffre.

Le second problème est le modèle de reconnaissance de chiffre. Nous avons décidé de créer un modèle très léger ne contenant pas beaucoup de couche. Un tel modèle décide plus rapidement mais se trompe aussi plus souvent… Pour pallier à ce problème, il faudrait revoir l'architecture du réseau de neurone.

Hormis ces deux problèmes, nous pensons que l'algorithme fonctionne plutôt correctement. En prenant en entrée l'image "sudoku.png" présent dans le dossier "images/raw/" la solution fonctionne très bien. Cela est dû au fait que le sudoku présent sur l'image a été écrit sur un ordinateur. Ainsi la luminosité de l'image ne laisse pas place au bruit et la clarté des chiffres est très élevée. Le réseau de neurone trouve donc très bien les chiffres présents dans les cases.

VIII. Perspectives

Il existe plusieurs concurrents majeurs à notre projet. Plusieurs applis mobiles sont spécialisées donc le scan et la résolution des sudoku. Par exemple, on peut citer Sudoku Solver – Scanner app using camera et Sudoku Solver Camera. Cette appli a un avantage considérable sur notre projet puisqu'elle permet de scanner sur un smartphone.



Nous avons également trouvé du code réalisant la même chose sur github. Le fait que celui-ci soit open source réduit fortement la possibilité de commercialisation de notre projet.
https://github.com/DiAnant/Sudoku-Scanner-And-Solver

Le seul avantage que notre projet a sur ces applications est qu'il est capable de lire des sudokus écrit à la main. Mais malheureusement c'est aussi le défaut de celui-ci car il a plus de mal à lire les chiffres imprimés. De plus, il est tout de même plus rare que les sudokus soient écrits à la main ; ce qui réduit le potentiel d'utilisation.

Enfin, les applications mobiles existantes sont très pratiques. Si nous souhaitions réellement le commercialiser, il faudrait améliorer l'interface utilisateur ainsi que la précision sur les nombres imprimés. Avec plus de temps nous aurions voulu utiliser la webcam de l'ordinateur et de superposer directement la solution au sudoku scanné en direct. Ceci pourrait être plus intéressant et nous démarquer des concurrents pour une potentielle commercialisation.

A la base, nous n'avons pas eu pour but de commercialiser ce projet, nous souhaitions réaliser un projet qui nous attirait et qui avait le potentiel pédagogique pour notre apprentissage. Nous nous sommes relevé un défi et sommes fiers du résultat malgré les quelques difficultés rencontrées.