Hiver2022

Résoudre un sudoku par capture d'image

ALEX SAVOIE
A00188646

INFO4304 | VISION PAR ORDINATEUR

Table des matières

1. Lien vers la vidéo présentation sur YouTube

https://youtu.be/b4wTh_9VLWQ

2. Introduction

Résoudre un sudoku est une tâche qui peut prendre quelques minutes où mêmes quelques heures à résoudre. Ne serait-il pas plus intéressant et rapide d'utiliser la puissance de calcul d'un ordinateur plutôt qu'un cerveau humain pour résoudre un sudoku? Dans ce projet, nous essayerons de résoudre une grille sudoku incomplète seulement par capture d'image. Ce projet consiste à transformer une image d'un sudoku incomplète en une image avec la solution. L'objectif principale du projet est de faire en sorte que le programme puisse fonctionner de manière générale et qu'il puisse résoudre n'importe quelle image d'un sudoku.

3. Base de données – Images

Pour ce qui est des images des sudoku, celles-ci sont tiré d'un livre intitulé 'Entraînez votre cerveau!'. J'ai utilisé deux caméras de téléphone portable différent dans le but d'essayer de voir si les algorithmes réussissent à bien fonctionner peu importe la caméra utilisée. Ces deux caméras sont celle du Samsung J3 Prime et celle du Samsung A8. Les images obtenues par les deux caméras contiennent des légères différences.

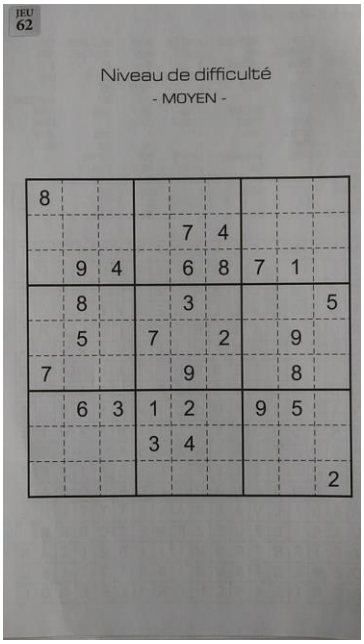


Image provenant du Samsung A8

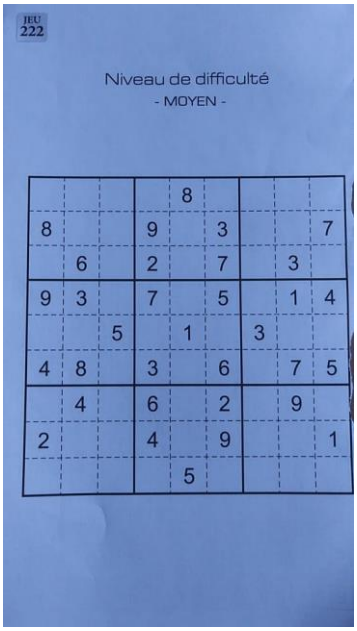


Image provenant du Samsung J3 Prime

Au total, notre base de données contient 5 images différentes. Pour ce qui est de l'éclairage, rien de trop spécial n'a été appliqué. Lors de la capture des images, le livre était tout simplement posé sur un bureau en utilisant la lumière ambiante de la chambre. Le flash des appareils était plutôt inutile et il est resté éteints lors de la prise des photos. J'ai aussi tenté de prendre des images de sudoku avec des perspectives différente, comme dans l'image ci-dessous, où l'image est plus courbée que souhaité :

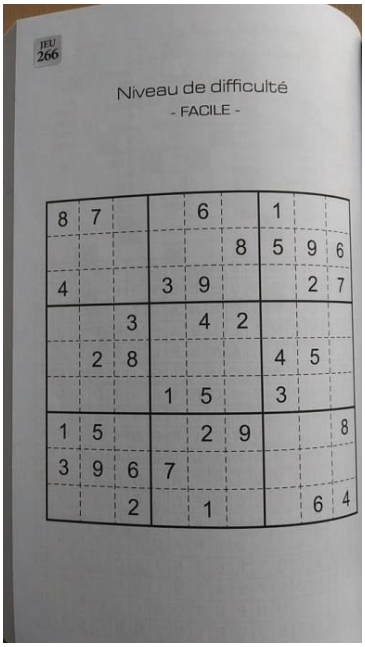


Image du sudoku avec une page courbée

Dans la prochaine section, nous verrons que nous aurons besoins de créer un réseau neurones convolutif (CNN). Pour ce faire, j'ai utilisé la très populaire *'mnist handwritten digit database[1]'* comme banque de données. Celle-ci sera utile pour la reconnaissance des chiffres dans nos grilles. Les explications viendront plus tard.

4. Algorithmes

Pour obtenir un résultat concret, nos images devront passer par plusieurs étapes de traitement. Le travail a été divisé en 7 sous-sections.

4.1 Traitement d'image

Cette étape est cruciale pour nos calculs. Cette étape à comme but à ce que tous nos images, peu importe la caméra utilisée, se ressemble à la fin du traitement. La première étape était de convertir nos images à une résolution de 960x540 (ratio 16:9). Inutile d'utilisé des résolutions originale car les calculs sont plus coûteux en termes de temps et les images avec des plus petites résolution fonctionne autant bien. Par la suite, les images sont converties en '*grayscale*'. J'ai ensuite ajouté un filtre gaussien de taille 5x5 avec un écart-type de 1 pour s'assurer qu'il n'ait pas de bruit sur les images. Et puis finalement, un 'adaptive threshold' est appliqué pour que l'image soit convertie en noir et blanc uniquement.

4.2 Bordure

Une fois l'image prétraité, il fallait appliquer un algorithme qui nous permettais de trouver toutes les bordures de l'image. Plus précisément, on était intéressé a trouvé la plus grande bordure de l'image, soit celle de la grille sudoku comme tel. Pour ce faire, il fallait créer une fonction qui nous permettait de trouver cette bordure. Cette fonction regarde chacune des bordures trouvées, une par une, et si l'aire est inférieure à 50pixels, on les ignore tout simplement. Par la suite, pour de celles qu'ils restent, on conserve celles qui contiennent exactement 4 coins et qui possède l'aire maximale car, en théorie, le plus grand carrée/rectangle de l'image devrait être notre grille sudoku. Finalement, notre fonction retourne les coordonnées des 4 coins de la bordure.

4.3 Arrangement des coins et transformation d'image

Il faut ensuite s'assurer que les coordonnées des 4 coins soient dans le bon ordre. Le coin en haut a gauche devrait être le coin '0', en haut à droite le coin '1', en bas à gauche le coin '2' et le coin en bas à droite le coin '3'. L'ordre des coins est important car cet ordre spécifique nous permet d'effectuer notre transformation d'image. On souhaite faire un genre de 'zoom' sur l'image pour seulement conserver la grille sudoku et éliminer tout ce qui se retrouve autour. Nous utiliserons les fonctions getPerspectiveTransform et warpPerspective de OpenCV. Notre nouvelle image aura une taille de 450x450.

4.4 Division des grilles

Par la suite, on souhaite diviser notre image en 81 nouvelles images correspondant à chacun des éléments de notre grille sudoku. Puisque notre image a une taille de 450x450, on peut facilement diviser notre grilles 9 fois à la verticale et 9 fois à l'horizontale pour obtenir 81 images de tailles 50x50. Ces images contiennent soit un chiffre entre 1 et 9 en blanc, ou une image complètement noire pour signifier que la case est vide. Ces 81 images sont placées dans un tableau où ils seront par la suite passé dans un réseau neurones convolutif.

4.5 Entraînement d'un modèle CNN et prédictions

Comme mentionner la section base de données, pour faire la reconnaissance des chiffres dans les cases, nous utiliseront la base de données *mnist handwritten digit database*. Notre réseau neurones convolutif est plutôt court et simple. La première couche est une couche convolutives avec 32 filtres avec une taille de fenêtre de 3x3. Ensuite, une couche de *'Max Pooling 2D*' est appliqué. En puis finalement, des couches 'Denses' sont ajouté pour faire la classification. Ce modèle est inspiré de l'article *'How to Develop a CNN for MNIST Handwritten Digit Classification[2]'*. Ce modèle nous retournait une liste avec les chiffres correspondant de chacune des cases. Les cases vides étaient représentées par des '0'.

4.6 Trouver la solution

Avec notre grilles sudoku dans une tableau, on doit maintenant trouver la solution. Puisque cette section n'est pas réellement pertinente pour le cours de INFO4304 – Vision par ordinateur, et que je ne voulait pas trop m'attarder sur cette partie du travail, j'ai utilisé une partie de code[3] retrouvé en ligne. Le code utilise le 'backtracking' pour résoudre rapidement une grille sudoku incomplète. Cet algorithme remplace les '0' dans notre tableau par les valeurs correspondante de la solution. S'il n'existe pas de solution, notre solutionneur de sudoku retourne la grille initiale avec des 0.

4.7 Affichage de la solution

La dernière étape est d'afficher le résultat final. On doit afficher seulement les chiffres où il avait des cases vides initialement. Donc, tous les cases qui contenaient un chiffre initialement restent intactes. Une nouvelle image est créée avec les chiffres manquant (en vert) placé aux endroits où se retrouverait normalement les cases vides. Cette nouvelle image subie, par la suite, une transformation d'image exactement comme dans la section 4.3, mais dans le sens inverse (*getPerspectiveTransform* et *warpPerspective*). Cette image est ensuite superposée à l'image originale de taille 960x540 pour afficher le résultat final.

5. Tests expérimentaux

Bien que l'on souhaite que notre programme fonctionne peu importe l'image de sudoku présenté, notre programme présente quand même quelques restrictions. Par exemple, notre programme est incapable de résoudre un sudoku si son image est orientée dans la mauvaise

direction. Le programme n'est pas en mesure de corriger automatiquement la mauvaise orientation et il assume tout simplement que l'image est déjà dans la bonne orientation. Donc, il est obligatoire que l'image soumise sois dans la bonne direction. Ensuite, le programme éprouve des difficultés lorsque quelqu'un a commencé à résoudre le sudoku au crayon ou à la plume. Le problème survient au niveau de la reconnaissance des chiffres de notre modèle CNN. Il arrive parfois que notre modèle est incapable de lire et comprendre l'écriture humaine. C'est pourquoi il est grandement préférable de soumettre un sudoku incomplet sans aucun écriture humaine. Aussi, une autre des restrictions connues est que l'image originale devrait avoir une résolution avec un ratio 16:9. Si cela n'est pas respecté, il est possible que lors des étapes de transformation d'image, l'image aurait une allure 'stretched'. Encore une fois, cela peut causer un problème au niveau de notre modèle CNN lors de la reconnaissance des chiffres si les chiffres sont étirés. Et puis finalement, notre programme a de la difficulté lorsque la grille de notre sudoku n'est pas un carré/rectangle presque parfait. Par exemple, dans l'image ci-dessous, la page du sudoku est courbée ce qui cause une certaine distorsion de la grille sudoku. Lorsqu'on essaie de découper l'image en 81 nouvelles images représentant chacune des cases, certaines des images sont mal découpées. En voici l'exemple avoir le '1' de la première rangée :

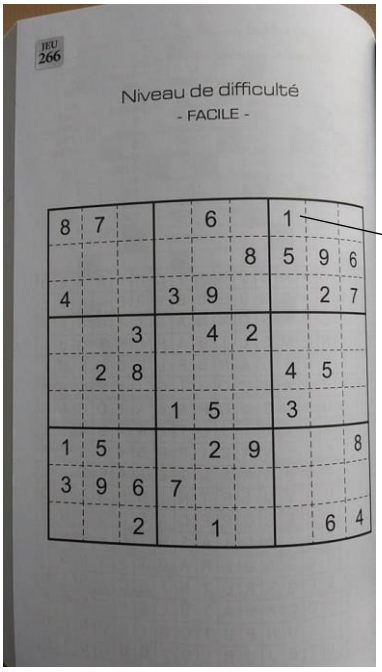
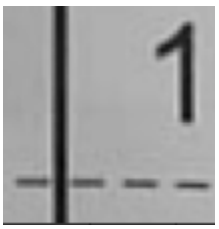


Image du sudoku avec courbure



Problème au niveau du découpage

Il a fallu continuellement apporter des légers changements à notre code pour que nos algorithmes fonctionnent le mieux possible. Dans la première partie, le prétraitement de l'image, il faut parfois appliquer un filtre gaussien pour réduire le bruit. Cela était le cas, surtout, pour des caméras qu'ils ne prennent pas des belles images. Le bruit dans l'image pouvait mener à une reconnaissance des chiffres erroné au niveau de notre modèle réseau neurone convolutif.

Un autre problème survenait lors du découpage de notre image sudoku en 81 nouvelles images contenant chacune des cases. Certaines des images contenait des bordures noires vers l'extérieur de l'image (voir image ci-dessous). La présence de cette bordure causait, encore une fois, un problème à la reconnaissance par notre modèle. Pour résoudre ce problème, j'ai tout simplement éliminé quelques pixels situés vers les extrémités des images pour que cette bordure soit enlevée de l'image.

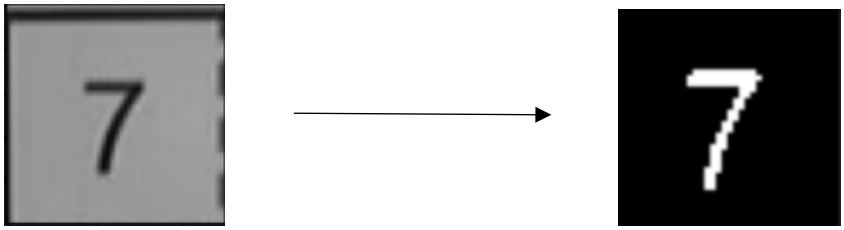


Image d'une case avec bordure non-désirée

Image de la case après la correction

6. Résultats

Dans cette section, nous allons analyser les résultats obtenus. Pour commencer, dans l'étape de prétraitement, nous avons diminuer la résolution de l'image, appliquer un filtre gaussien ainsi que transformer l'image en noir et blanc grâce a la méthode adaptiveThreshold de OpenCV.

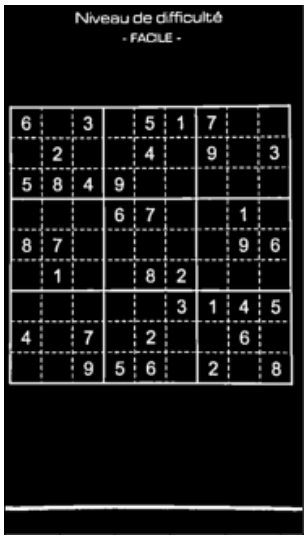


Image du sudoku ayant subis les étapes de prétraitement

L’image obtenu à la suite du traitement affiche une grille sudoku qui est claire et compréhensible. Les chiffres seront facilement reconnaissables par le CNN. Par la suite, il fallait trouver toutes les bordures de l’image et ensuite conserver celle ayant l’aire maximale et ayant exactement 4 coins.

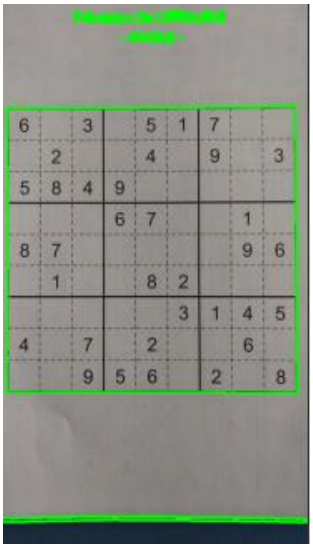


Image du sudoku avec les bordures en vert

Une fois cette bordure principale, les coordonnées des quatre coins sont conservées pour effectuer une transformation d’image lors de la prochaine étape. Il fallait s’assurer que les coins sont dans un ordre prédéfini. Lorsque nous ferrons le ‘zoom’ sur l’image, l’ordre des coins est très important. Par exemple, le coin en haut à gauche, le coin 0, devrait être celui ayant une coordonnée en x minimale avec un coordonnées en y maximale.

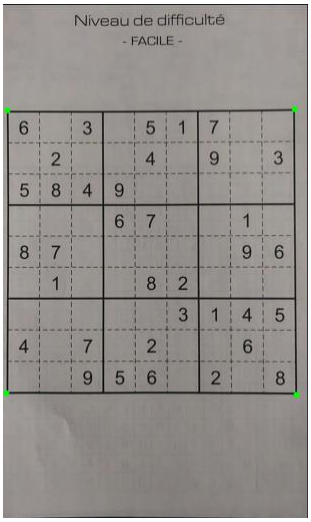


Image du sudoku avec les coins en vert

Une fois les quatre coins bien ordonnés, on ‘zoom’ sur le carré former par ces coins. Cette image résultante aura une dimension de 450x450.

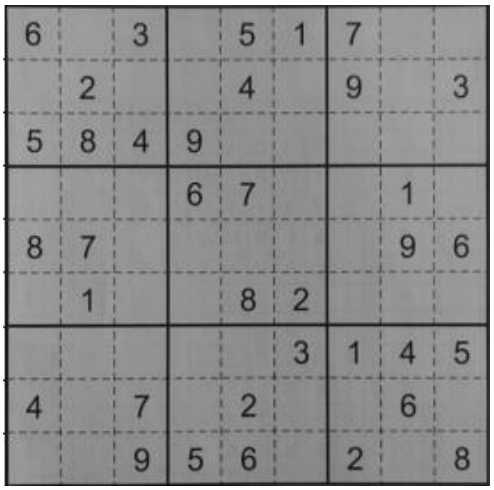


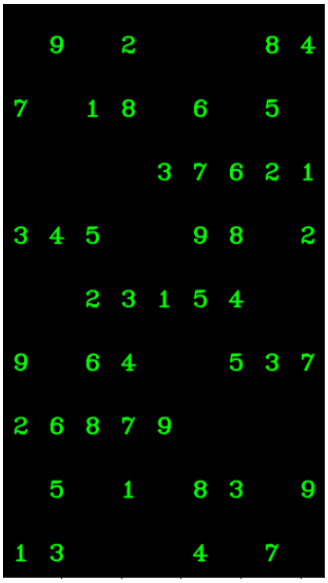
Image une fois le ‘zoom’ effectué

Image du ‘7’ de la première rangée après le découpage

Ensuite, cette image sera découpée 9 fois à la verticale et 9 fois à l’horizontale pour former 81 nouvelles images correspondant à chacune des cases de la grilles.

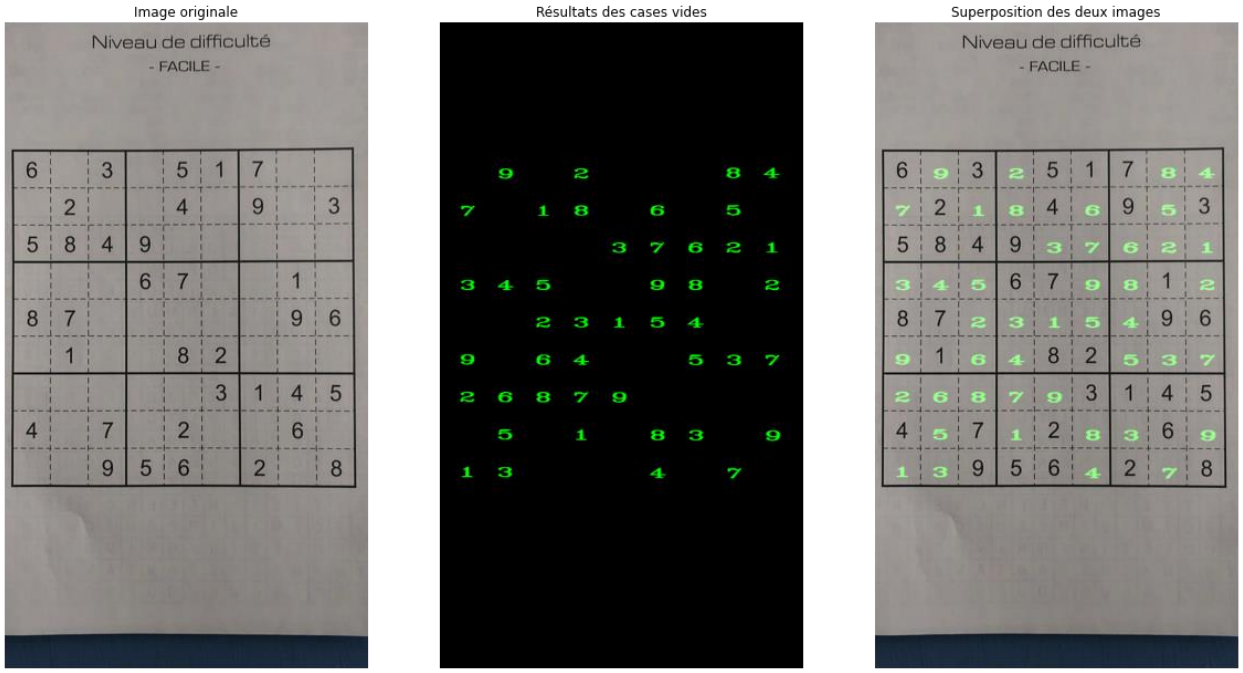
Par la suite, ces 81 images seront placées dans notre modèle CNN pour être analysé. Notre modèle CNN est peu profond et comporte très peu de couche mais il arrive toutefois à nous donner une exactitude de 98,75% avec les données d’entraînement de notre base de données ‘mnist’. Dans la grande majorité des cas, notre modèle ne rencontrait aucune difficulté à faire la prédiction les chiffres dans notre grille.

Une fois la reconnaissance des chiffres effectué, il ne reste plus qu’à trouver la solution de notre sudoku et d’afficher la réponse au-dessus de l’image originale. L’algorithmes utilisé pour résoudre notre sudoku fonctionne rapidement. Dans la plupart des cas, il réussit à résoudre le sudoku en moins d’une seconde. Les chiffres manquants trouvé par notre algorithme sont placer dans une nouvelle image. Cette image sera par la suite superposé sur l’image originale pour obtenir le résultat final.



Chiffres de la réponse où les cases sont vides

Rendu a ce point, il ne reste plus qu'à transformer cette image avec les chiffres en vert pour qu'elle puisse être superposer à l'image originale. Pour ce faire, il suffit de faire les mêmes transformations lorsque nous avons fait le zoom sur l'image mais cette fois, les étapes de la transformation seront faites dans le sens inverse. Voici à ce que le résultat final ressemble une fois les images superposées.



Le résultat final correspond à la superposition de l'image avec les chiffres en vert et l'image originale

7. Conclusion

Malgré les petits problèmes découverts tout au long du projet, la résolution des grilles sudoku fonctionnaient très bien dans la grande majorité des cas. Le résultat obtenu semble être assez fiable pour conclure que les objectifs du projet ont été atteints avec succès.

8. Références

[1] LeCun et al.. The MNIST Database of handwritten digits. The Courant Institute of Mathematical Sciences NYU. November 1998. MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges

[2] Jason Brownlee. How to Develop a CNN for MNIST Handwritten Digit Classification. Deep Learning for Computer Vision (Machine Learning Mastery). May 2019. How to Develop a CNN for MNIST Handwritten Digit Classification (machinelearningmastery.com)

[3] TechWithTim. Python Solver with Backtracking. June 2019. Python Sudoku Solver w/ Backtracking - techwithtim.net