Sudoku Solver - Final Report

Alessandra Guimarães Oestreich 8531953
Jhonathan Roberto Viúdes 8532001

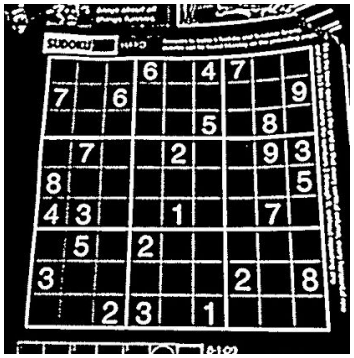
---

1. Objective

The objective of this project is to identify a Sudoku Puzzle of an input and pre-process it in order to obtain only the puzzle, creating an undistorted image.
A grid presented in a Sudoku is known as a standard 9x9 matrix. Taking in consideration that the image has its perspective adjusted, the image is cropped and each one of the boxes and its contents are individually stored.
In order to solve the puzzle, the program has to identify the digits from each box. Analyzing the contents of the boxes previously stored, the program generate the matrix containing the puzzle and the same is solved using a backtracking algorithm.

2. Steps

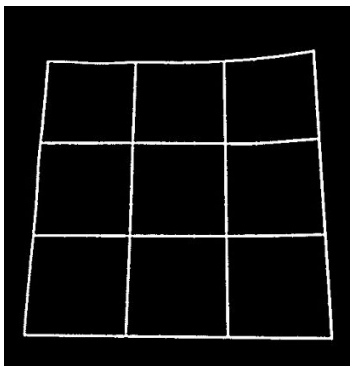
2.1. Subdivide the puzzle



The original image of the puzzle is pre-processed so we can get only the dark regions of the input image.
This pre-processing involves steps in order to obtain the desired result:

- Guassian Blur: smooths the noise of the original image;
- Threshold: selects the black parts of the image;
- Dilate: since threshold can disconnect the lines of the puzzle, dilating the image fills the spaces that may have appeared.

2.2. Detecting the grid



Assuming that the sudoku grid is the biggest object in the image, the algorithm find the biggest bounding box and save its location with white lines (the outerbox) as shown in the image.

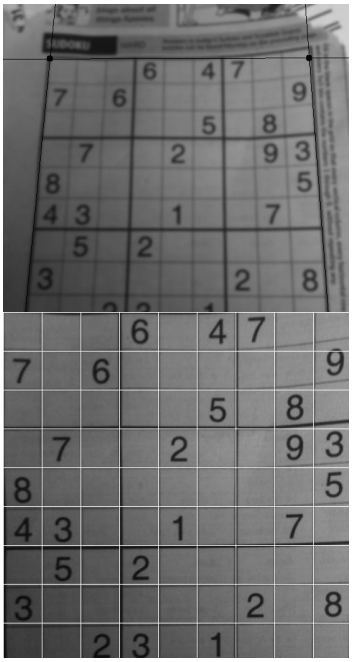
This step is important so we can identify the shape of the current puzzle and find the lines that best describe it. Still we have the internal lines and possible external lines interfering in the result.

2.3. Locate the puzzle

To find the better lines and their positions the Hough transformation is used. It returns the lines of the image in mathematical terms in order to know their exact position in the image and not only its white pixels.
However, the transformation returns inumerous lines and not all of them are needed, so to simplify the lines, we merge the ones that are closer to each other.
Analyzing the remaining ones, it is known that the edges needed for the WrapPerspective are those nearest to the edges of the image. Since we have the position of the lines, the program is able to find the intersections and return the four corners of the puzzle as coordinates.

2.4. Fixing the perspective

Using the WrapPerspective function and the coordinates, the program adjust the original image and stretches the puzzle itself so the borders of it are the borders of the image as well.



2.5. Crop boxes

After adjusting the puzzle and identifying the grid, the lines found in the previous steps are used to overlap the original grid so it won't affect the digit recognition.
The next step is to divide the puzzle into a 9x9 grid. Each cell in the grid will correspond to a box on the puzzle and each one of them is saved as an new image.

2.6. Identify the numbers

In order to solve the puzzle it is necessary to know which box holds which digit. The program access the image of each cell and recognizes the image using the pytessaract library and store the information in an integer matrix.

2.7. Solve the puzzle

Using a backtracking algorithm the program is able to solve the puzzle contained in the matrix and return the result in the terminal.

3. Results

Using the previous specified steps, the program is able to identify and solve the puzzles. However, its accuracy relies mostly on the quality of the image and the font used in it.
Pytesseract holds an image to string function, which was used with the right configuration and image treatment to have better results.

3.1. Instructions

To run the program with an specific image, move it to the same folder as the main program (FinalProject.py) and run the code with the image name as an argument.

python FinalProject.py <image>

4. Problems

4.1. Accuracy

The accuracy of the program relies in many factor and the result is not always the desired one.

4.1.1. Original image quality

Depending on board's distortion, the results of the wrap and crop can affect the result of the board because of the boxes that may be out of place, resulting on the non recognition of the numbers in them.

4.2. Digit recognition

Initially a neural network was used to identify the numbers using the LinearSVC but the results were not as expected and another solution had to be found.
The digit recognition with pytesseract sometimes can't identify certain number of the boxes but if this happens it is considered as an empty box and when the board is solved this digit not always interfere in the solution.

5. Conclusion

The project was a challenge because each image has its own peculiarities that can make it harder to the program to treat the flaws and solve it. As expected, not all of the images have success in the process and some are not solvable.