Image to Sudoku Solver

Tomáš Motus
ČVUT–FIT
motustom@fit.cvut.cz

June 1, 2025

1 Goals

1.1 Basic solver

The main goal was to create a Sudoku solver that was fast and easy to use, unlike many other Sudoku solvers you can find online.

1.2 Detect from image

The idea behind it is to use some kind of image recognition to find and extract Sudoku into a usable format.

1.3 Usable on phone

Also it would be nice if it could be used on a phone, so try to implement a direct camera feed.

2 Interface

The whole interface was created using a Python package called Streamlit. It handles image taking and uploading images, an interactive and modern looking GUI, light and dark mode, displaying Sudokus and inputs, and running backend code based on user input.

3 Input

The interface needs to be as versatile as possible.

3.1 Direct text

The first step was to create a simple Sudoku table in which the user can write numbers. The position of the numbers should be accessible by the backend, which can then compute the solved Sudoku.

3.2 Uploading an image

3.2.1 Past vs present ideas

The first idea that came to mind was to use an OpenCV to detect Sudoku squares and divide them into cells. These cells would be sent to some kind of OCR(optical character recognition) model, which would be pre-trained or trained by myself to detect numbers in the cell and store it. After some experimentation, it proved to be unreliable even when using images with pretty good quality. Upon searching for different OCR models that would fix recognition issues, I found a model that changed my plan.

3.2.2 Detecting a Sudoku in the image

The first step was to detect where in the image the Sudoku is and crop the image to that location. It happened to be not as easy to do reliably as I thought. I created a whole pipeline, which now works pretty well, even if the Sudoku is tilted or with bad resolution.

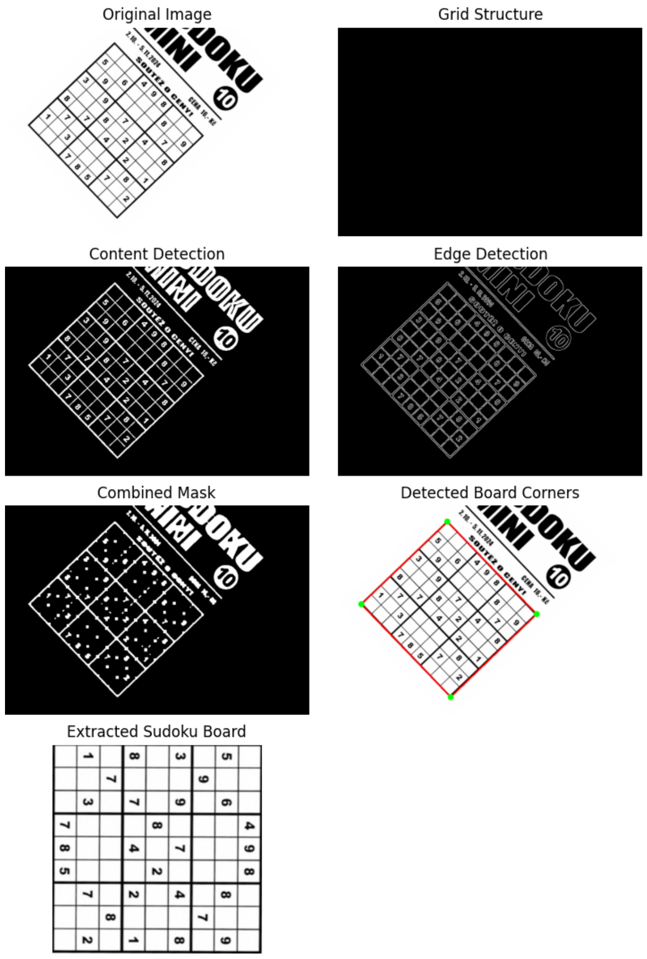

Figure 1: Detection pipeline of rotated image

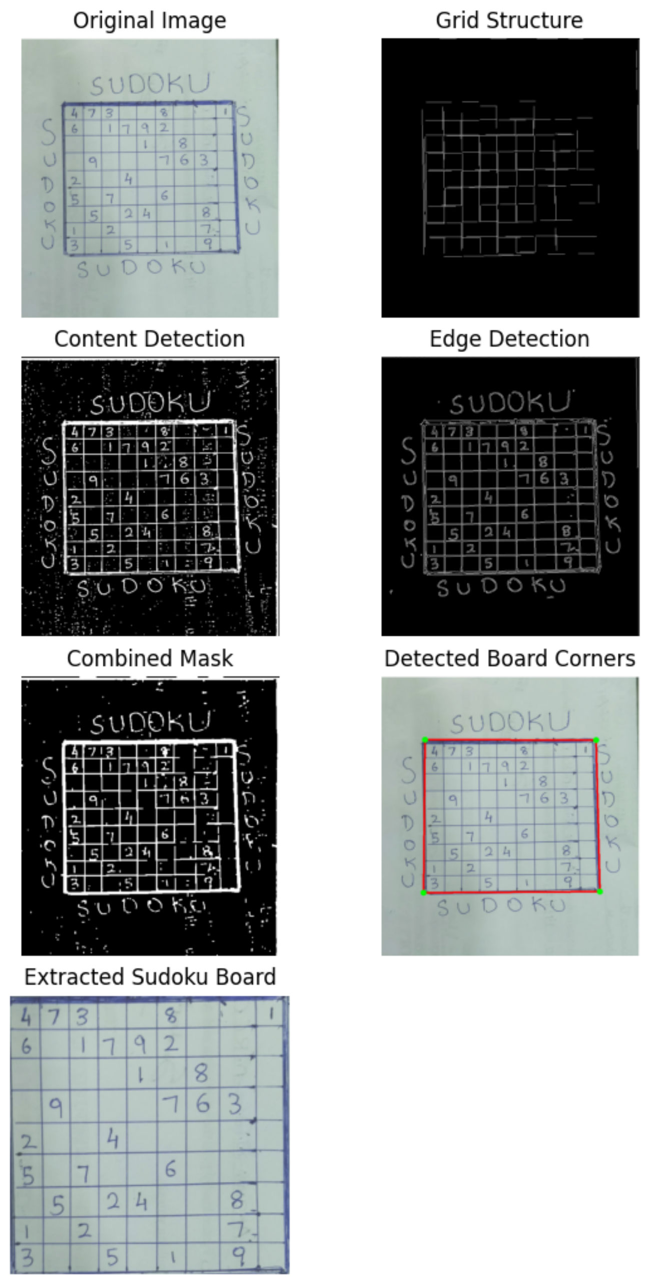

Figure 2: Detection pipeline of handwritten Sudoku

3.2.3 Using OCR

While exploring different approaches, I found a Python library named PaddleOCR. It is based on the paddlepaddle library and has robust features. One of the best that I used was good text recognition and accurate text detection with coordinates with respect to the image. Thanks to image detection's cropping, I was able to use the midpoint of the detected number as a position in the Sudoku board.

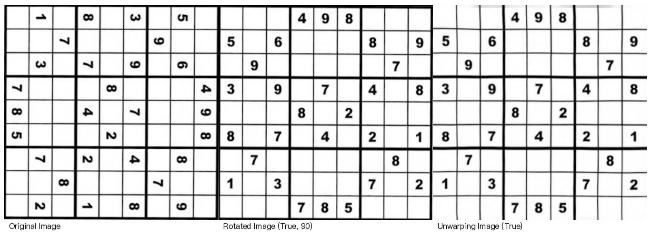

Figure 3: Preprocessing for OCR

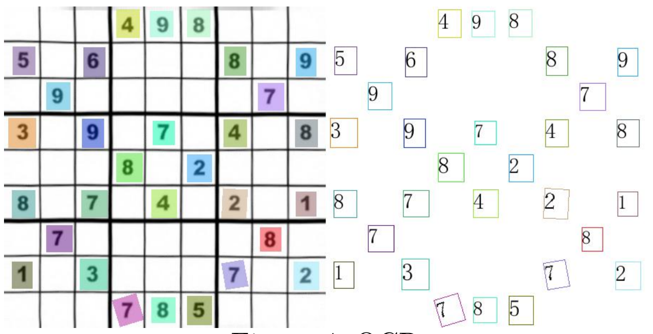
Figure 4: OCR

3.2.4 Filtering artifacts

When the OCR encounters low-resolution, low-quality images, it usually finds some artifacts that need to be filtered. I filtered them based on a few things:

1. If the detected char was not a number
2. If the box was significantly smaller than expected based on cell dimensions

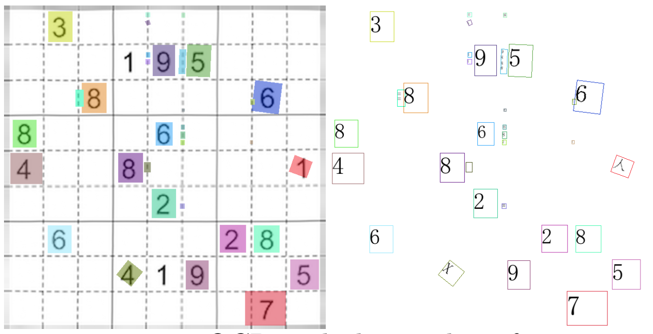
Figure 5: OCR with detected artifacts

3.3 Taking a photo

Taking a photo was a pretty easy task to implement. The basic functionality was already in Streamlit, which I extracted, saved as a temporary file, and passed the saved image into the already existing image-upload pipeline.

3.4 View detected Sudoku

In GUI there is a drop-down that allows the user to see what they have uploaded and how it was processed.

3.5 Editing detected Sudoku

Input Sudoku allows the user to edit the Sudoku in case of bad recognition of chars(for example: 1 instead of 7) or bad Sudoku detection.

4 Solving the Sudoku

Until the last few commits, the Sudoku solver was a simple basic implementation that solved it by backtracking. Later, it evolved into something bigger because of speed.

4.1 Making faster Sudoku solver

It needed to be faster because the program automatically solves immediately after the new number is inserted.

4.1.1 Logic solve

First step was to do a "Logic solve". This means that the program solves it like a human would:

1. Find places where only one number can go
2. Find only place where a number can go in given unit(row, column, grid)

4.1.2 MRV backtracking

After partially filling the Sudoku using a fast logic solve comes the backtracking using MRV(minimum remaining values). The program has a list of candidates for each cell. Based on that, it starts the backtracking in the cell that has the least candidates, so there is a big probability that the picked number is the right number, which decreases the average time complexity.

5 Output

The output is displayed the same as the input, but it does not allow the user to type in the Sudoku. The base state is an empty Sudoku, and after the first entry of the digit in the input Sudoku, the solution is found and displayed in the output.

6 User error handling

I implemented two error messages:

1. Warning that the Sudoku user had written or was recognized is not valid and has some kind of mistake(two same numbers in a unit)
2. Warning that the Sudoku is impossible to solve, which the only way to tell is to try to solve it.

7 Testing and code quality

7.1 PEP8

The code is mostly created according to PEP8. Most of the violations of PEP8 are lines too long by a bit, which are not really important, and there are a lot of local variables and nested block due to a lot of pre-computing and backtracking.

7.2 Pytest

The part of the code that performs the computation is mostly covered by Pystests, which test the basic functionality of every function. The part which is GUI is not tested at all, because the easiest test to do is to just try how it looks, thanks to the interactive part of Streamlit.

8 Results

In the end, I created a pretty functional Python program that can be set up on a server to be accessible as a webpage to all kinds of user. It can be used pretty fast and is not that hard to run.

9 Conclusion

I also tried to set up the Sudoku solver as a community server on Streamlit, but because of PaddleOCR's dependency paddlepaddle the image part of the program does not work because paddlepaddle tries to additionally install things into its folder(venv) when it is first started(using ___init___.py), which is not allowed on the server(venv is located in home of adminuser).
The link to the app is:
https://crazyaisudokusolverultraedition.streamlit.app/

References

[1] Anthropic. Claude sonnet 4. online, 2025. Help with code part and debbuging [cit. 2025–5–27] https://claude.ai.

[2] OpenCV-Open Computer Vision Library. Opencv documentation. online, 2025. [cit. 2025–5–27] https://docs.opencv.org/4.11.0/index.html.

[3] paddlepaddle and paddleocr comunity. Paddleocr. online, 2025. [cit. 2025–5–27] https://github.com/PaddlePaddle/PaddleOCR.