Sudoku OCR Journey Documentation

Introduction

The Sudoku OCR (Optical Character Recognition) journey involves extracting a Sudoku puzzle grid from an image and solving it using computer vision techniques. The goal is to develop an automated system that can read and solve Sudoku puzzles captured in images.

Software Used

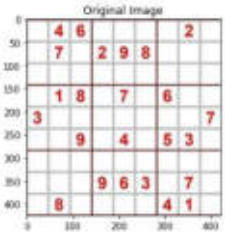
- OpenCV: A computer vision library used for image preprocessing, line extraction, noise removal, and other image manipulation tasks.
- Tesseract OCR: An open-source OCR engine used for character recognition.
- NumPy: A numerical computing library used for handling grid data structures and array operations.
- Matplotlib: A plotting library used for visualizing intermediate and final images.
- PIL (Python Imaging Library): A library used for opening, manipulating, and saving images.

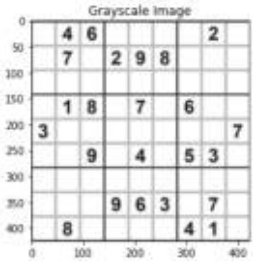
Objectives

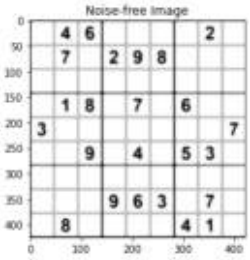
1. Preprocess the image to isolate the Sudoku puzzle grid and enhance digit readability.
2. Extract the individual cells of the Sudoku grid.
3. Perform OCR on each cell to recognize the digits present.
4. Solve the Sudoku puzzle using a backtracking algorithm.
5. Allow user interaction to edit the Sudoku grid before solving.

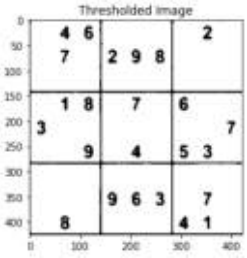
Image Preprocessing

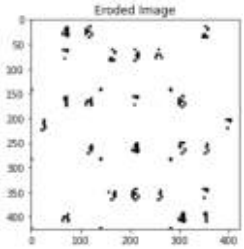
General functions and their effects:

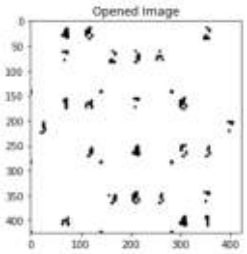


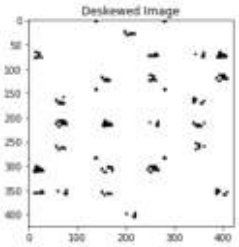


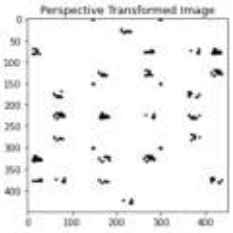












Preprocessing steps used:

1. **Grayscale Conversion**
   The image is converted from RGB to grayscale using the **cv2.cvtColor()** function from OpenCV. Grayscale conversion simplifies further processing by reducing the image to a single channel representing intensity values.

2. **Binarization**
   The grayscale image is binarized using adaptive thresholding with the **cv2.adaptiveThreshold()** function. This step separates the foreground (digits) from the background, resulting in a binary image.

3. **Line Extraction**
   Hough Transform, implemented through the **cv2.HoughLinesP()** function, is utilized to detect lines present in the binary image. This step helps identify the grid lines of the Sudoku puzzle.

4. **Line Removal**
   Detected lines from the previous step are removed from the binary image using the **cv2.line()** function. This process eliminates the grid lines, isolating the Sudoku digits for further processing.

5. **Noise Removal**
   To reduce noise and smoothen the image, a denoising filter (median blur) is applied using the **cv2.medianBlur()** function. This step improves the accuracy of digit extraction and subsequent OCR.

6. **Erosion**
   Erosion is performed using the **cv2.erode()** function to enhance the structure and readability of the Sudoku digits. It helps improve the segmentation of individual digits, making them more distinguishable.

7. **Cell Extraction**
   The eroded image is split into individual cells representing each digit of the Sudoku puzzle. The **np.vsplit()** and **np.hsplit()** functions from NumPy are used to divide the image into a 9x9 grid, resulting in 81 individual cells.

OCR and Sudoku Solving

OCR (Optical Character Recognition)

For each cell, the OCR process is performed using the Tesseract OCR engine. The **pytesseract.image_to_string()** function is used to extract text from each cell image. A custom OCR configuration is set to recognize only digits and exclude other characters.

Sudoku Puzzle Solving

The Sudoku puzzle grid is represented as a 2D list of digits. Initially, the OCR results are used to populate the grid with the recognized digits. Then, a backtracking algorithm is applied to solve the Sudoku puzzle. The algorithm finds an empty cell, tries numbers from 1 to 9, and recursively solves the puzzle until a solution is found or all possibilities are exhausted.

Editing Sudoku

The system allows users to edit the Sudoku grid before solving. Users can specify the row number and column numbers to modify, with '0' representing a blank cell. This feature enables interactive interaction and customization of the Sudoku puzzle.

Challenges Faced

1. Divisibility of Cells: One challenge encountered was ensuring that the Sudoku grid cells were correctly extracted. Resizing the image and adjusting the parameters for splitting the cells helped overcome this challenge.

2. Misreading of Digits: Another challenge was the misreading of certain digits, especially the digit '1' being mistaken for '4'. Training the OCR model specifically for Sudoku digits could help improve accuracy, but for simplicity, a basic approach was adopted.

3. Distinct Boundaries: Maintaining distinct boundaries between the cells while removing grid lines was crucial. However, this led to some double-digit numbers being misread. Striking a balance

between distinct boundaries and accurate OCR results required experimentation and fine-tuning.

Conclusion

The Sudoku OCR journey encompasses various image preprocessing techniques, OCR using Tesseract, Sudoku solving algorithms, and user interaction. By combining computer vision and machine learning, the system provides an automated way to extract and solve Sudoku puzzles captured in images. The integration of OpenCV, Tesseract OCR, and other libraries offers a powerful framework for building Sudoku OCR applications and exploring other image-based puzzle-solving tasks.

Also, after a LONG time of trial and error yes, the order of the preprocessing steps can affect the final result. The order of the preprocessing steps for success are:

1. **Grayscale conversion**: Convert the image to grayscale.
2. **Binarization**: Apply adaptive thresholding to convert the grayscale image to a binary image.
3. **Line removal**: Use Hough Transform to detect and remove the lines from the binary image.
4. **Denoising**: Apply a denoising filter (e.g., median blur) to reduce noise in the image.
5. **Erosion**: Perform erosion to further enhance the structure and readability of the digit

   There is a factor of the extent of each that need to be tested too to get optimal results for all photos.

Robustness Test:

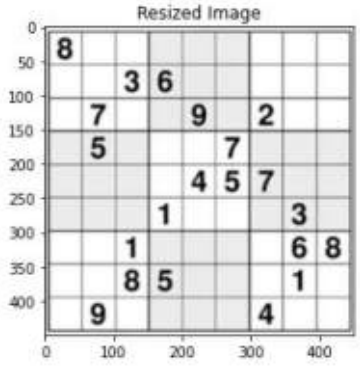
Current Sudoku Grid:
OCR Results:
8 0 0 0 0 0 0 0 0
0 0 3 6 0 0 0 0 0
0 7 0 0 9 0 2 0 0
0 5 0 0 0 7 0 0 0
0 0 0 0 4 5 7 0 0
0 0 0 0 0 0 0 3 0
0 0 1 0 0 0 0 6 8
0 0 8 5 0 0 0 1 0
0 9 0 0 0 0 4 0 0
Enter the row number to edit (1-9), or enter '0' to solve the puzzle: 6

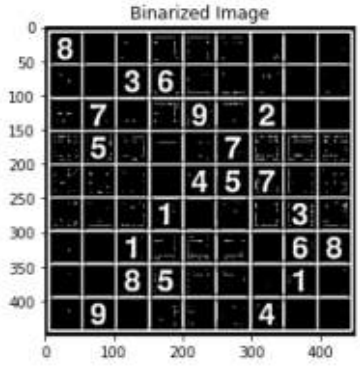
Enter the column numbers to edit (1-9), separated by spaces, with zero to represent gaps: 0 0 0 1 0 0 0 3 0

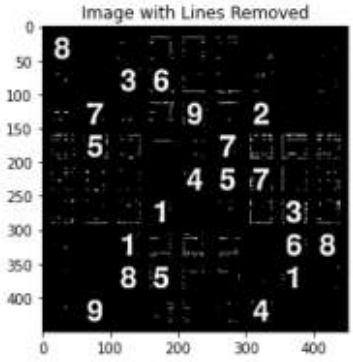
Current Sudoku Grid:
Current Sudoku Grid:
OCR Results:
8 0 0 0 0 0 0 0 0
0 0 3 6 0 0 0 0 0
0 7 0 0 9 0 2 0 0
0 5 0 0 0 7 0 0 0
0 0 0 0 4 5 7 0 0
0 0 0 1 0 0 0 3 0
0 0 1 0 0 0 0 6 8
0 0 8 5 0 0 0 1 0
0 9 0 0 0 0 4 0 0

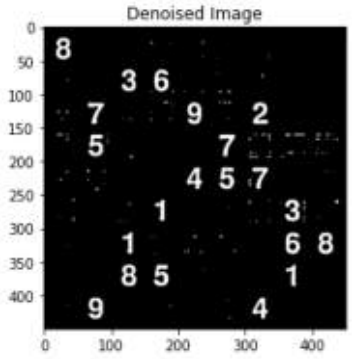
Enter the row number to edit (1-9), or enter '0' to solve the puzzle: 0

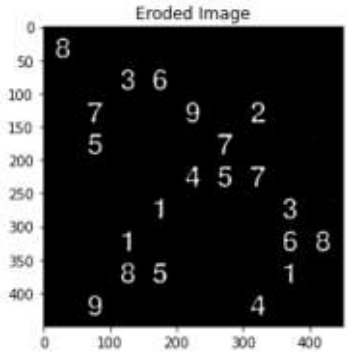
Answer:
Current Sudoku Grid:
OCR Results:
8 1 2 7 5 3 6 4 9
9 4 3 6 8 2 1 7 5
6 7 5 4 9 1 2 8 3
1 5 4 2 3 7 8 9 6
3 6 9 8 4 5 7 2 1
2 8 7 1 6 9 5 3 4
5 2 1 9 7 4 3 6 8
4 3 8 5 2 6 9 1 7
7 9 6 3 1 8 4 5 2











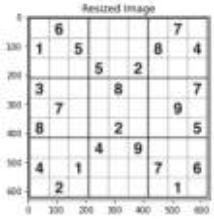
Answer:
Current Sudoku Grid:
OCR Results:
8 1 2 7 5 3 6 4 9
9 4 3 6 8 2 1 7 5
6 7 5 4 9 1 2 8 3
1 5 4 2 3 7 8 9 6
3 6 9 8 4 5 7 2 1
2 8 7 1 6 9 5 3 4
5 2 1 9 7 4 3 6 8
4 3 8 5 2 6 9 1 7
7 9 6 3 1 8 4 5 2

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

actual answer.

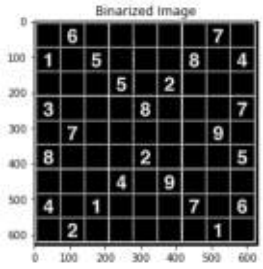
https://towardsdatascience.com/pre-processing-in-ocr-fc231c6035a7 preprocessing tips

Notes and Chicken scratch of journey

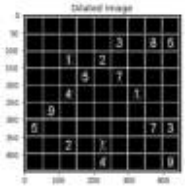


Issue was the cutting up of the square was not divisible. So resize.

Other issue now is that lines are being read as 1, and the recurved 1 is looking like a 4. It could do with training but for this model I want it to be simple.

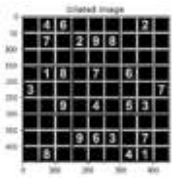


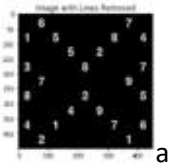
To help with distinct boundaries, but now there are more readings of double digit numbers.



0 4 6 0 0 0 0 2. 0
0 0 0 2 9 0 0 0 0
0 0 0 0 0 0 0 0 0
0 4 8 0 0 0 0 0 0
3 0 0 0 0 0 0 0 7
0 0 0 0 4 0 5 0 0
0 0 0 0 0 0 0 0 0
0 0 0 9 6 0 0 7 0
0 8 0 0 0 0 4 0 0

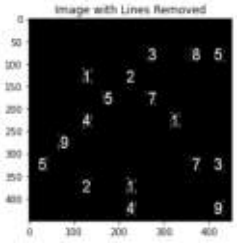
Almost, but not quite. Machine learning looks like it will have to be incorporated as 1s are often misread.





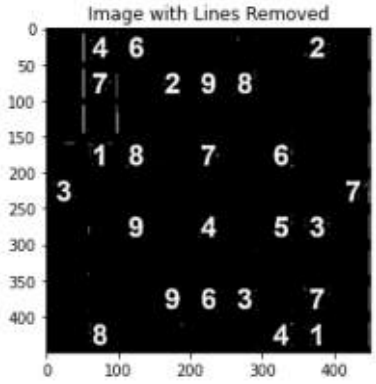
almost there!

0 6 0 0 0 0 0 7 0
1 0 5 0 0 0 8 0 4
0 0 0 0 0 2 0 0 0
0 0 0 0 8 0 0 0 7
0 7 0 0 0 0 0 9 0
8 0 0 0 2 0 0 0 5
0 0 0 4 0 9 0 0 0
4 0 1 0 0 0 7 0 6
0 2 0 0 0 0 0 1 0



really good except 1 is read as 4:

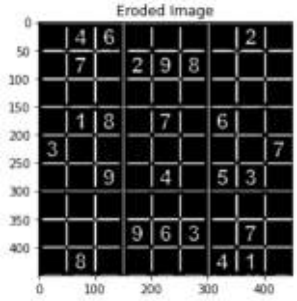
0 0 0 0 0 0 0 0 0
0 0 0 0 0 3 0 8 5
0 0 4 0 2 0 0 0 0
0 0 0 5 0 7 0 0 0
0 0 4 0 0 0 4 0 0
0 9 0 0 0 0 0 0 0
5 0 0 0 0 0 0 7 3
0 0 2 0 0 0 0 0 0
0 0 0 0 4 0 0 0 9



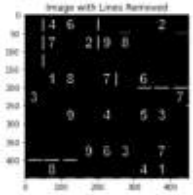
0 4 6 0 0 0 0 2 0
0 7 0 2 9 8 0 0 0
0 0 0 0 0 0 0 0 0
0 1 8 0 7 0 6 0 0
0 0 0 0 0 0 0 0 7
0 0 9 0 4 0 5 3 0
0 0 0 0 0 0 0 0 0
0 0 0 9 6 3 0 7 0
0 0 0 0 0 0 4 0 0
Missed a few on this one

Yes, the order of the preprocessing steps can affect the final result. In your desired sequence, the order of the preprocessing steps would be:

6. Grayscale conversion: Convert the image to grayscale.
7. Binarization: Apply adaptive thresholding to convert the grayscale image to a binary image.
8. Line removal: Use Hough Transform to detect and remove the lines from the binary image.
9. Denoising: Apply a denoising filter (e.g., median blur) to reduce noise in the image.
10. Erosion: Perform erosion to further enhance the structure and readability of the digit

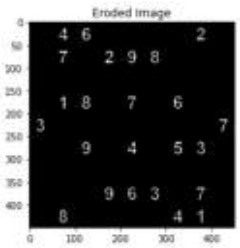


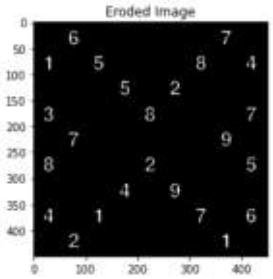
eroding lines doesn't make much sense here as we are left with ill defined horizontals leading to this:



SUCCESS

0 4 6 0 0 0 0 2 0
0 7 0 2 9 8 0 0 0
0 0 0 0 0 0 0 0 0
0 1 8 0 7 0 6 0 0
3 0 0 0 0 0 0 0 7
0 0 9 0 4 0 5 3 0
0 0 0 0 0 0 0 0 0
0 0 0 9 6 3 0 7 0
0 8 0 0 0 0 4 1 0





erosion factor can be relaxed but it Is doing really well!

0 6 0 0 0 0 0 7 0
1 0 5 0 0 0 8 0 4
0 0 0 5 0 2 0 0 0
3 0 0 0 8 0 0 0 7
0 7 0 0 0 0 0 9 0
8 0 0 0 2 0 0 0 5
0 0 0 4 0 9 0 0 0
4 0 1 0 0 0 7 0 6
0 2 0 0 0 0 0 1 0

Backtracking:

OCR Results:
0 6 0 0 0 0 0 7 0
1 0 5 0 0 0 8 0 4
0 0 0 5 0 2 0 0 0
3 0 0 0 8 0 0 0 7
0 7 0 0 0 0 0 9 0
8 0 0 0 2 0 0 0 5
0 0 0 4 0 9 0 0 0
4 0 1 0 0 0 7 0 6
0 2 0 0 0 0 0 1 0
Answer:

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