Solving Sudoku from an Image using Modular Architecture Approach

Manav B. Sanghavi, Aniket K. Rupani, Mahek S. Maniar, Sai Deepthi Pabba
Computer Department
K.J. Somaiya College of Engineering
Mumbai, India
Email: manav.s@somaiya.edu; aniket.r@somaiya.edu; mahek.maniar@somaiya.edu, psaideepthi@somaiya.edu

Abstract**—**Sudoku puzzles can be found in various physical forms in newspapers, magazines and elsewhere. It may often be desirable to convert this puzzle into a digital format for ease of solving. This paper proposes a method for extracting and solving a Sudoku puzzle captured in an image. AI techniques can then be applied to solve the Sudoku puzzle. A modular architecture is created for this purpose. Modules can be replaced as needed, making it easier to improve and maintain an application using the proposed architecture

Keywords*- sudoku; image processing; artificial intelligence; architecture; local thresholding; number detection; corner detection*

*****

I. Introduction

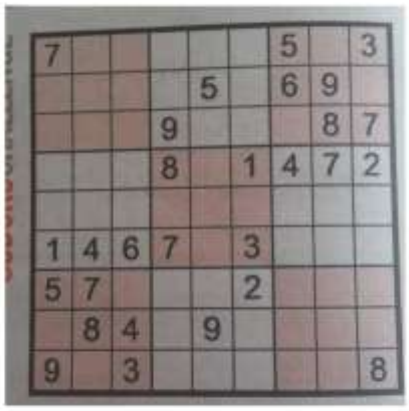
Sudoku is a very popular puzzle game. A typical Sudoku puzzle consists of a 9-by-9 grid made up of nine 3-by-3 subgrids. Digits appear in some squares and based on these starting “clues”, a player completes the grid so that each row, column and subgrid contains the digits 1 through 9 exactly once.

Sudoku puzzles are usually found in newspapers and magazines. However, more often than not, it is inconvenient to solve them as they are printed on paper, since it would not be possible to rewrite a digit once it has been written. Even using an eraser to remove a digit written with a pencil would ultimately result in degradation of the printed puzzle. Hence, it would be more convenient to simply take a picture of the Sudoku puzzle and convert it into a digital format, in order to solve it.

This paper details the steps which are used to detect and extract the Sudoku puzzle from an image. A modular architecture for this purpose is proposed. Since various different techniques are used in the process of extracting the Sudoku puzzle, a modular architecture allows for improvement and maintainability of an implementation of this process as individual module implementations can be changed more easily without affecting the rest of the software implementation.

Image processing components give us the digital grid of the Sudoku puzzle as output. This is stored as an appropriate data structure and then given to an AI module for solving the puzzle. Like the image processing steps, the AI component is modular. Hence, it is easy to swap out the algorithm used for solving the Sudoku puzzle or improve the AI implementation gradually, over time.

II. Problem Statement



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

Extracting the Sudoku puzzle from an image involves the following steps in general:

1. Preprocessing the image
2. Finding the Sudoku grid’s outer box
3. Finding the box corners
4. Using the box corners to extract cells
5. Detecting the number present in each cell
6. Creating a virtual Sudoku grid

Various image processing and computer vision techniques may be applied at each individual step. Preprocessing constitutes of resizing the image to an appropriate size, noise removal and thresholding [1]. Finding the outer Sudoku puzzle box involves region detection. Extracting box corners can be done by using filters for corner detection. These corners can then be used to extract the 81 cells of the image and then template matching can be done to find the value of the digit in non-empty cells.

Solving a Sudoku puzzle is an interesting area of research. This is because Sudoku is an NP-Complete problem [2]. The proposed architecture converts Sudoku puzzles into digital versions which reduces the tedium involved in creating multiple Sudoku grids by hand, when multiple Sudoku puzzles are required to carry out research into solving Sudoku puzzles or developing new and improving old algorithms for solving problems lying in the class of NP-Complete..

III. Solution Approach and Methodology

As we have seen in the previous section, extracting the Sudoku puzzle from an image follows a certain process. Therefore, it is possible to create a modular architecture for extracting the Sudoku puzzle from an image containing a Sudoku puzzle. This architecture is given in Figure 1.

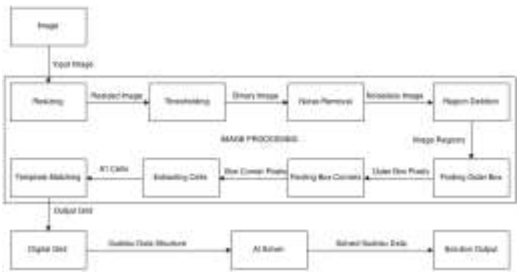
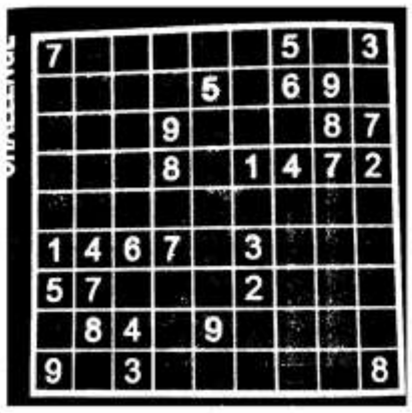


Figure 1

The steps as detailed in the Problem Statement are expanded upon in the following sections.

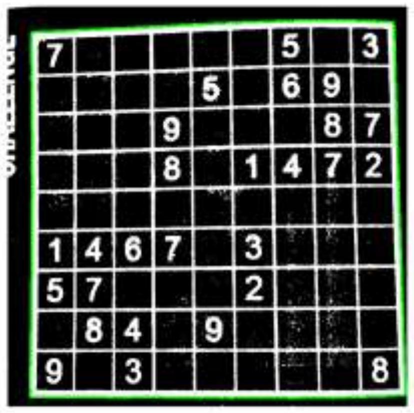
A. Preprocessing



The image is resized to some suitable dimensions. An image size of 480 x 480 is sufficient for further processing. The colour image has to be converted into a binary image. Objects of interest, like the Sudoku grid and digits, are to be labelled as object or foreground pixels and everything else is to be labelled as background. Local or global thresholding may be used for this purpose. Local thresholding is preferred over global as local thresholding is more resistant to strong illumination gradients or shadows [3].

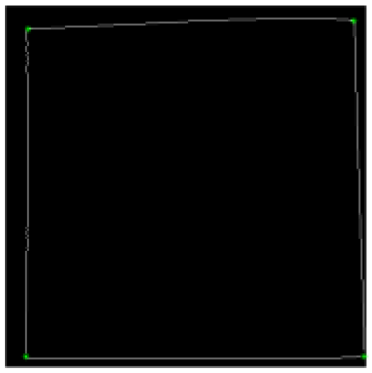
The captured image may have noise and so noise removal would be beneficial for extracting the correct information from the image. Median filter can be applied to remove the salt and pepper noise.

B. Finding the Sudoku grid's outer box



Region Detection can be applied to find the various regions in the image. For finding the image regions, edge detection may first be applied: this will result in regions having only boundary pixels. This would improve the results of the region detection. Out of all the regions, the outer Sudoku grid box would be the largest region. Therefore, by calculating the number of pixels constituting the border of the region and selecting the largest result would be one possibility for finding the outer box. Another approach could be to calculate the area of the regions and select the region with maximum area [1].

C. Finding box corners



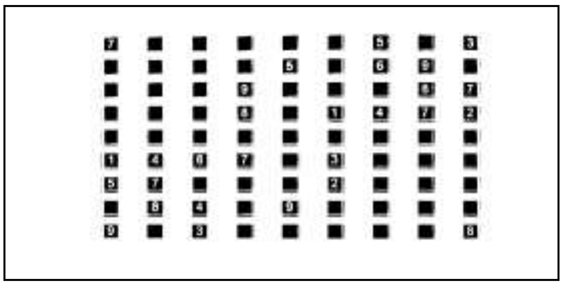
Once the Sudoku box is detected, its four corners need to be extracted. This is done so that the four corner pixels can be used as reference to extract the 81 cells of the Sudoku puzzle. Finding the box corners can be done using filters for detecting corners or by using corner detection techniques like Harris operator. However, due to inevitable skews induced in the image, the box may have multiple corner points detected. In this case, there may be a need to change the parameters of corner detection method used or use a different technique for corner detection. Alternatively, the algorithm developed by us can be used. This algorithm accepts a list of corner pixels detected on a box and selects the best four corner pixels possible.

Initially, the corner pixels are sorted in ascending according to their column values. Then, the list of corners is traversed row-wise to find the optimal corners. A simple textual representation for this algorithm to find the lower left corner is given as follows:

```
1. START
2. Assign first value as the upper left corner
3. Assign second value as the lower left corner
4. Loop through all the corner pixels
        i. If the current pixel is below the current lower
           left corner by a certain threshold, then go
           to step ii; else go to step iv
       ii. If the current pixel value is to the left of, or
           within the allowable tolerance limit, to
           the right of the current lower left corner,
           go to the next step; else go to step iv
      iii. If the distance between the upper left and
           current pixel is greater than the distance
           between the upper left and current lower
           left pixel, then set the current pixel as the
           new lower left pixel
       iv. If there are corners left, go to step i; else go
           to step 5
5. END
```

The above algorithm finds the lower left corner by taking reference of an estimated upper left corner. Using this value of the lower left corner, the accurate value of the upper left corner can be found. Similarly, by looping backwards appropriately, we can find the upper right and lower right corner pixels.

D. Using the box corners to extract cells



Once the four corners of the box have been detected, these four corner values can be used to divide the Sudoku box into 9 x 9 cells. The advantage of this approach is that it is easier to find out which cell is placed where in the grid. Many of the grids would be empty; hence it would be advantageous to apply template matching on non-empty cells. Finding the average of all pixel values inside a cell and taking those cells whose average values are above a certain threshold would be one method of selecting non-empty cells. However, if grid lines are present inside an extracted cell, this would induce errors. A better method would be to only consider the average of the pixel values inside a central window so as to not allow the grid lines lying at the edge of cells to induce errors.

E. Detecting the numbers present in each cell

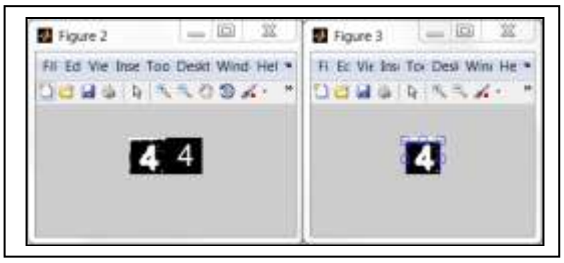
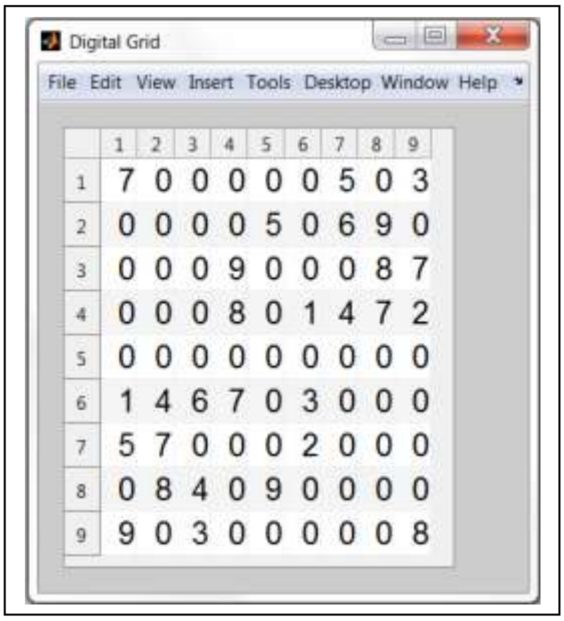


Figure 2
Fil Ed Vie Inse Too Deskt Wind Hel
4 4
Figure 3
Fi Ec Vie Ins Toc Desi Win He
4

Once we have extracted the cells from the image, we can apply template matching on non-empty cells. The digits inside the cells may be skewed due to improper placement of the camera. Hence, robust template matching is required [4]. A set of templates may be made as per the requirement. Multiple sets of different templates fonts can also be used. If multiple template sets are to be used, then template matching may need to be applied on each and every cell using each and every set. However, this may be avoided if while template matching, a particularly high confidence value is encountered for a particular digit, allowing us to recognize that particular digit. In this case, we may proceed with only that template set for the entire image and abandon processing using other template sets.

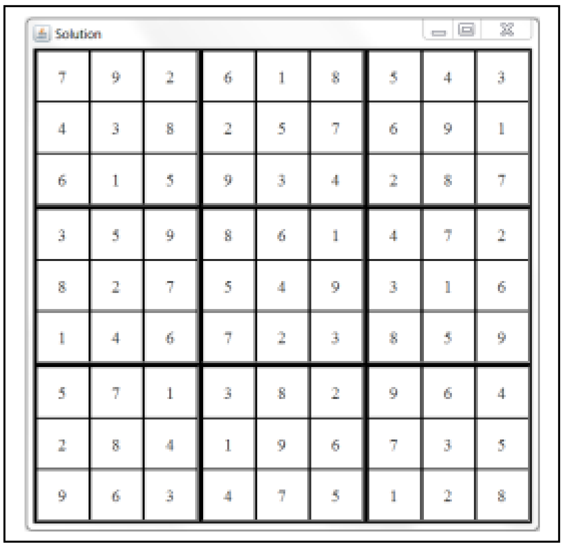
F. Creating a digital Sudoku grid



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

Once all the digits have been identified, a data structure has to be made to store the Sudoku grid. A two dimensional array may be used to store the values conveniently, with zeroes representing blank cells. Alternate representations can also be implemented for particular cases, e.g. storing it as a single dimensional array may be faster for an AI or a Sudoku solving algorithm's implementation to solve the puzzle [6].

G. *Solving the Sudoku puzzle*



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

Once all the digits have been identified and an appropriate data structure has been made, this is passed to the AI component for solving. Sudoku is traditionally solved as a Constraint Satisfaction Problem (CSP) using Backtracking technique.

A novel mothed for solving Sudoku by using Backtracking is considered for use [5]. Instead of backtracking over all 81 cells, permutations are made for all subgrids and CSP is applied for these subgrids, in a predefined order. This reduces redundant calculations, thus speeding up overall solving process of the puzzle.

IV. Conclusion

We have proposed a modular architecture for solving a Sudoku puzzle from an image. The advantage of following this architecture is its modularity: if a module's implementation needs to be improved, it can be changed relatively easily, as long as the input to the module and output from the module remains unchanged. Such an architectural model would be useful for developing applications which work on Sudoku puzzles, as the modular nature allows for quickly customizing the nature of the output supplied by simply changing parameters or modules of the architecture.

Furthermore, the Sudoku solving component can also be replaced with different modules, new algorithms can be rapidly integrated into a software implementation following this architecture and by modifying appropriate modules the proposed architecture could also be used for researching and solving NP-Complete problems.

V. Further Work

We now aim to find a concrete and efficient implementation of the proposed architecture as well as apply and compare various Sudoku solving techniques to find an optimal Sudoku solver for the proposed architecture.

Acknowledgment

We would like to extend our sincerest gratitude to our esteemed mentor for this project, Prof. Sai Deepthi Pabba.

References

[1] Simha, P.J.; Suraj, K.V.; Ahobala, T.; "Recognition of numbers and position using image processing techniques for solving Sudoku Puzzles", Advances in Engineering, Science and Management (ICAESM), 2012 International Conference on Year: 2012 ; Pages: 1-5

[2] Aaronson, L.; "Sudoku Science", Spectrum, IEEE (Volume:43, Issue: 2), Year: 2006, Pages: 16-17

[3] B. Kim ; D. Park; "Adaptive image normalisation based on block processing for enhancement of fingerprint image"; Electronics Letters (Volume: 38, Issue: 14), Year: 2002 ; Pages: 696-698

[4] W. Xu ; X. Huang ; X. Li ; Y. Zhann ; J. Zhang ; W. Zhang; "An affine invariant interest point and region detector based on Gabor filters", Control Automation Robotics & Vision (ICARCV), 2010 11th International Conference on, Year: 2010, pages: 878-883

[5] Maji, A.K.; Pal, R.K.; "Sudoku solver using minigrid based backtracking", Advance Computing Conference (IACC), 2014 IEEE International, Year: 2014, Pges: 36-44

Web References

[6] Norvig, P.; "Solving Every Sudoku Puzzle", http://norvig.com/sudoku.html, Last Accessed on: 1st April, 201