Processing Sudoku Images and Solving by Recursive Backtracking Algorithm

Mrs. Supriya H.S[1] Ayush Baran[2] Abhishek Nipun[3] Kanhaiya Kumar[4] Parul Prabhat[5]
[1]Assistant Professor [2,3,4,5]Student
[1,2,3,4,5]Department of Computer Science & Engineering
[1,2,3,4,5]Sir MVIT, India

Abstract— We propose a method of detecting and recognizing the elements of a Sudoku puzzle and providing a digital copy of the solution for it. The method involves a image-processed Sudoku Solver. The solver is capable of solving a Sudoku directly from an image captured from any digital camera. This would mainly require 5 steps. Step 1 recognises the grid of the puzzle. Step 2 extracts the grid of the puzzle. Step 3 is the extraction of the numbers. Step 4 recognises the numbers extracted. Step 5 solves the puzzle by recursive backtracking algorithm. The digits are recognised by the OCR method Artificial Neural networks.

Key words: Processing Sudoku Images, Recursive Backtracking Algorithm

I. INTRODUCTION

In real life we come across Sudoku puzzles of varying difficulty levels in newspapers and other text and digital media. It is a common leisure activity for a lot of people. However, it is observed that the solution is not always immediately available for the user's verification. In most cases, people have to wait till the next day to check the solutions of the Sudoku they just solved. Hence our motivation for this project was to develop an application on an android device for this purpose. In our application, the user needs to capture a clean image of an unsolved Sudoku Puzzle, which then returns the complete solution of the same.

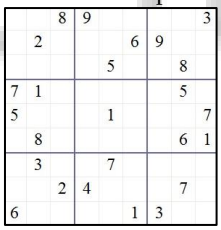

Fig. 1: A sample Sudoku Puzzle

Sudoku is a logic based puzzle with the goal to complete a 9x9 grid so that each row, each column and each of the nine 3x3 boxes contain the numbers 1 through 9, given a partial filling to a unique solution. The problem itself is a popular brainteaser but can easily be used as an algorithmic problem, with similarities to the graph colouring problem only with fixed size and dependencies.

The modern version of Sudoku was introduced in Japan in the eighties and has since then attracted a lot of programmers and therefore there are a great deal of different Sudoku solving algorithms available on the Internet, both implementations and construction methods. Sudoku solvers are interesting because they are practical applications of very theoretical problems, and most people are familiar with them.

II. EXISTING SYSTEM

Currently, Sudoku puzzles are becoming increasingly popular among the people all over the world. Today, the game appears in almost every newspaper, in books and in many websites. It is a common leisure activity for a lot of people. However, it is observed that the solution is not always immediately available for the user's verification. In most cases, people have to wait till the next day to check the solutions of the Sudoku they just solved.

III. PROPOSED SYSTEM

Our motivation for this project was to develop software for this purpose. In our application, the user needs to capture a clean image of an unsolved Sudoku Puzzle, which then returns the complete solution of the same after applying appropriate pre-processing to the acquired image.

A. Processing Sudoku images and solving by Recursive Backtracking Algorithm

Suppose we are given a Sudoku puzzle on a piece of paper and we have to find a solution of that puzzle. The task here would be to create software to analyse the puzzle and give proper output.

Our approach consists mainly of 5 steps-

1) Step 1- Grid Detection

For the grid detection, the image should be in a greyscale so that it doesn't cause many problems during the thresholding. If it's not, we have to turn the image into greyscale which can be done by calculating the mean of the red, blue, green values of a pixel and then setting RGB values of the pixels as the same.

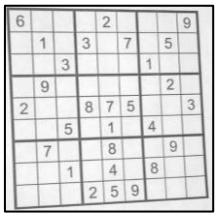

Fig. 2: Greyscale image

Now for the pre-processing of the image, first we have to blur the image so that noise can be avoided. After that we have the find the threshold of the image. The image can have varying illumination levels, so a good choice for a thresholding algorithm would be adaptive threshold. It calculates level several small windows in the image. This threshold level is calculated using the mean level in the window. So it keeps things illumination independent.

Since we're interested in the borders, and they are black, we invert the image outer Box. Then, the borders of the puzzles are white (along with other noise).

Now we have to find the biggest BLoB. For this First, We use the flood fill command. This command returns a bounding rectangle of the pixels it filled. We've assumed the biggest thing in the picture to be the puzzle. So the biggest blob should have be the puzzle. Flood Fill is done by assigning unique labels to the connected components of the

image and then finding the blob with the largest number of pixels with that label.

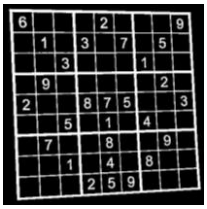
Fig. 3: Threshold image

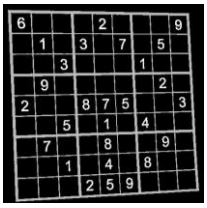
Fig. 4: Found biggest Blob

2) Step 2- Grid Extraction

After Finding the BLoB, We have to detect the corners of the grid. This is done by checking the neighbours of the pixels with that label and checking if it can be a corner or not.

Suppose we have to find the upper left corner. For this, we check that it's lower and right pixels are white and none of the pixels in the top left portion of the image from that pixel contain any pixel of that label. If there is a pixel with that condition true, it will be our upper left corner.

Similarily, we can find all the four corners.

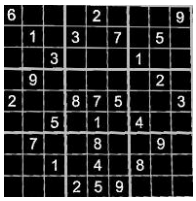
Fig. 5: Grid Extracted

After finding all the four corners of the grid, we have to skew and crop the image so that all the noise and extra blobs be removed.

3) Step 3- Digit Extraction

Since the grid is of the puzzle of a 9x9 Sudoku what we can do is to divide by the image by $1/9^{th}$ of the height and width respectively and save them as separate images. But before this we have to make sure the cells that we are extracting does not contain any noise. For this, we remove the boundaries of the image and only keep the numbers with a different label than the grid.

On doing this we can get all the cells of the puzzle as a different image.

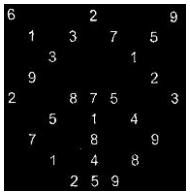
Fig. 6: Boundaries Removed


Fig. 7: Extracted Cells

4) Step 4- Digit Recognition

Since we have got all the cells in different images, what we have to do now is to scan all the images and perform OCR to recognise what number is in which cell.

To do this, we will be using the OCR method artificial neural networks. Here we store a collection of different types of Handwriting for all the numbers and on encountering a certain pattern of pixels in the image, we compare the pattern with the training data that we have collected beforehand. By comparing we make a smart guess to find the number that is in the image. If a number is found we store that in its appropriate location and go to the next cell's image.

To avoid redundancy, we can check if the density of pixels of the cell's image is very less. If it is very less then the image may not be having any digits inside it and may not be checked as it may be redundant to do OCR on an empty image having no digits inside it.

5) Step 5: Solving by Recursive Backtracking Algorithm

Like all other Backtracking problems, we can solve Sudoku by one by one assigning numbers to empty cells. Before assigning a number, we check whether it is safe to assign. We basically check that the same number is not present in current row, current column and current 3X3 sub grid. After checking for safety, we assign the number, and recursively check whether this assignment leads to a solution or not. If the assignment doesn't lead to a solution, then we try next number for current empty cell. And if none of number (1 to 9) lead to solution, we return false.

B. Algorithm

Find row, col of an unassigned cell

If there is none, return true

For digits from 1 to 9

- If there is no conflict for digit at row, col assign digit to row, col and recursively try fill in rest of grid
- If recursion successful, return true
- Else, remove digit and try another

If all digits have been tried and nothing worked, return false

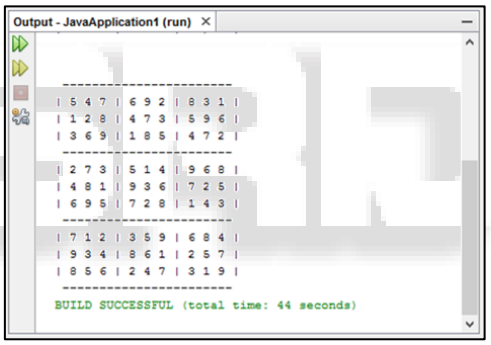

Fig. 8: Output of the puzzle

IV. Conclusion

In this paper, we developed a method to process Sudoku puzzle's images and produce their solution. After applying appropriate pre-processing to the acquired image we use efficient area calculation techniques to recognize the enclosing box of the puzzle. A virtual grid is then created to identify the digit positions .Tesseract OCR (optical character recognition) engine is used as a method for digit recognition. The actual solution is computed using a backtracking algorithm. Experiments conducted on various types of Sudoku questions demonstrate the efficiency and robustness of our proposed approaches in real-world scenarios. The algorithm is found to be capable of handling cases of translation, perspective, illumination gradient, scaling, and background clutter.

References

[1] Otsu, N., "A Threshold Selection Method from Gray-Level Histograms," IEEE Transactions on Systems, Man, and Cybernetics," Vol. 9, No. 1, 1979, pp. 62-66.

[2] "An Efficient and Versatile Flood Fill Algorithm for Raster Scan Displays" by C. Bond, 2011.
[3] "Burns, Robert."C. Harris and M. Stephens (1988). "A combined corner and edge detector". Proceedings of the 4th Alvey Vision Conference. pp. 147151,"
[4] "Optical Character Recognition using Neural Networks", by Deepayan Sarkar, University Of Wisconsin- Madison ECE 539 Project, Fall 2003.
[5] "An Overview of the Tesseract OCR Engine", Proc. Ninth Int. Conference on Document Analysis and Recognition (ICDAR), IEEE Computer Society (2007), pp. 629-633. Publication year 2007.
[6] S.V. Rice, G. Nagy, T.A. Nartker, Optical Character Recognition: An Illustrated Guide to the Frontier, Kluwer Academic Publishers, USA 1999, pp. 57-60.
[7] "A report on the Sudoku Solver", at Bachelor's Essay at dept. Computer Science, Royal Institute of Technology Spring 2013. Pg 13.

Web References

[8] https://www.tutorialspoint.com/java_dip/grayscale_conversion.htm
[9] http://www.javaxt.com/documentation/?jar=javaxt-core&package=javaxt.io& class=Jar
[10] http://aishack.in/tutorials/sudoku-grabber-opencv-detection/

