Mobile SuDoKu Harvesting App

Benjamin Zwiener
Department of Computer Science
Doane University
1014 Boswell Ave, Crete, NE, 68333
benjamin.zwiener@doane.edu

Abstract

The purpose of this project was to create an Android application that uses computer vision to harvest the layout and published difficulty of unsolved SuDoKu puzzles, and then upload this data to a database. The harvested data will be used to train a neural network, designed by another student, with the purpose of automatically classifying the difficulty of Sudoku puzzles.

Sudoku is a logic game with very simple rules. You have a 9x9 grid where the goal is to put a digit from 1-9 in each row, column, and sub-3x3 grid. Even though the rules are very simple, the game can be quite complex to solve. There are many techniques to solving a Sudoku puzzle so determining a difficulty can be very challenging.

Keywords: Computer vision, Sudoku, mobile app

1 Introduction

1.1 Background history

Sudoku is a widely played game across many age groups. Sudoku is highly touted for its easy to understand rules but hard to solve puzzles. Sudoku is credited to Howard Garnes who worked for the American Dell Magazine. He introduced the puzzle in 1979 as a modified version of Leonhard Euler's Latin Squares and the historic Magic Squares. It was first called Number Place. The game became very popular in Japan where it was renamed to Sudoku (Japanese) or Number Single (English). Su translates to number and Doku translates to Single.

The creation of Sudoku has created many intriguing mathematical and computer science problems to be solved. Most, if not all, Sudoku puzzles are now generated by computers. This in itself is a fun problem to solve as a game must have only one solution. Another mathematical problem involving Sudoku is how many possible puzzles are there. In a 2005 paper by Bertram Felgenhauer called "Enumerating possible Sudoku grids," there is an estimated 5,472,730,538 unique Sudoku puzzles. This was solved with a computer program and later verified. The last problem I will bring up involving Sudoku puzzles is that of classifying its difficulty.

1.2 Origins of project

Doane's Computer Science Department has an ongoing SuDoKu project. The project already has a puzzle generator that submits puzzles to the University's Doane Owl. Another student is now training a neural network to determine the difficulty of a Sudoku puzzle. To test the neural network, some other application that would read in Sudoku puzzles to turn them into an 81 character string – representing the puzzle – while also recording the published difficulty was needed.

This jumped out as a computer vision problem. It came about at the right time as Doane University has a three-year Digital Imaging and Vision Applications in Science (DIVAS) program to teach computer vision techniques to undergraduate students. DIVAS introduces natural science undergraduate students to Python programming and computer vision techniques. In May 2017, the DIVAS scholars participated in a five-day programming boot camp, where we learned the fundamentals of bash, git, Python, and the OpenCV computer vision libraries. After the boot camp, we worked on genuine research projects, both in pairs and individually. The SuDoKu harvester was part of our individual projects.

2 Initial Plans

The first rough draft of the project envisioned a mobile Android application where the user would snap a picture of a Sudoku puzzle, let a computer vision algorithm using OpenCV determine the character – if any – in each square, verify the algorithm did the correct identification, and then classify the puzzles published difficulty. All of this information would then be sent to a database to be tested.

This would require the creator to learn enough Android application development to access the smart phones camera capabilities and then send the taken images to a Java language OpenCV algorithm as Android uses the Java programming language. Understanding the differences between OpenCV's Java and Python libraries was also needed as the DIVAS boot camp only taught the Python library. Finally, being able to understand how to send the data to a hosted database was needed.

3 Setbacks

Right away there were setbacks with the project. The developing environment for Android is enormous with several APIs and versions. After choosing an initial API, we came along the problem that the online tutorials only work for specific versions of Android and there are little to no comments in their code. This brought about a long continuing setback as the learning curve is steep. Because of this, most of the time has been spent creating a Python prototype to process the puzzles. The algorithm for this can then be transferred over to Java.

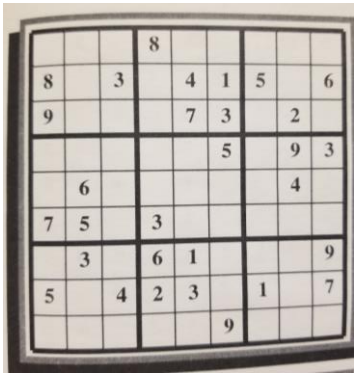

Figure 1. Sudoku puzzle used for testing.

3.1 Attempt one

Another setback is figuring out a way to classify the digits on the grid. The first algorithm we attempted to implement was using basic OpenCV functions. Some of the basic functions were turning a colored RGB image to its grayscale equivalent, blurring an image using the Gaussian blur technique, using a threshold function to create a binary image, and finally a find contours function. Contours of an image are the boundaries to objects. The boundaries are found from a binary image through quick changes from black to white. The first steps were used on images of digits 1-9. Grayscale, blur, and make the image binary

- Find the contours on the image
- Store each contour in a list

These contours will later be used to compare shapes with other contours.

Figure 2. Image used for digit one.

Figure 3. Image used for digit 2.

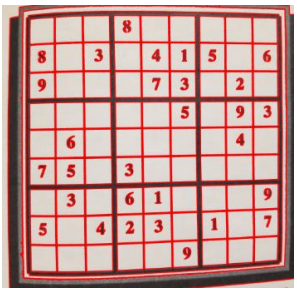

Figure 4. All contours found.

The next steps were done on an image of a Sudoku puzzle.

- Grayscale, blur, and make the image binary
- Find the contours on the image
- Loop through the list of contours and use a function called *matchShapes()* to determine if the contour is a digit or not

What is done can be seen in Figures 2, 3 and 4. There were several problems. First was too many contours were found so it took a long time to go through all the contours. The attempted solution for this was finding a threshold number for the contour area. OpenCV has a function that determines the area called *contourArea()*. This was also a pain because it never found all the digits correctly. We tried for several days get this to work, but with no real success. We ended up deciding that even if we could solve it for this puzzle, there would be many problems with other versions of Sudoku that used different fonts. This brought us to our second attempt.

3.2 Attempt two (houghlines)

For the second attempt at solving the classification problem we decided to attempt to cut up the grid first to separate all the squares. At this point we had decided we would try to classify the digits using some type of neural net – which will be talked about in Section 4 Current Work. After looking for solutions on the internet, we found what looked like an easy process that implemented OpenCV. The found solution, using hough lines, can be seen in Figure 5.

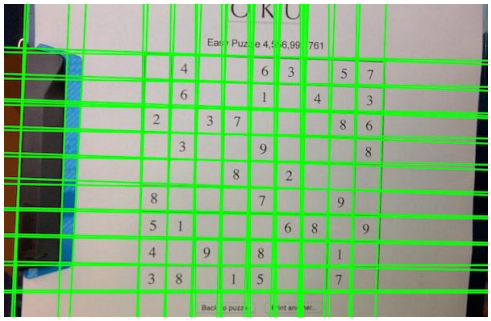

Figure 5. Solution found at http://jponttuset.cat/solving-sudokus-like-a-pro-1/.

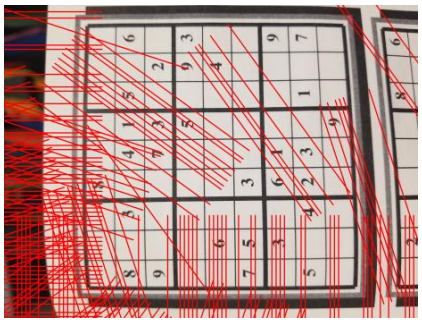

Figure 6. Our best attempt at finding the hough lines.

The solution found on the internet looked simple and straightforward, but did not work out the way it should have. After hours of making small changes, Figure 6 shows the best attempt at finding the hough lines. We assume the problem had to do with either the bold outline of the puzzle or something on the background. In current work, we are still toying with this method to try to get it to work while trying out a different method.

4 Current Work

Our Professor Mark Meysenburg at Doane University is trying to put together a camera application tutorial to help us learn the specifics needed for Android. While he is doing that, we are just trying to put together an algorithm to separate each square from the grid. The reason we are only focusing on separating the grid is because we found an easy way to classify the digits.

We were able to train a neural net how to classify digits using deep learning techniques. This was done at the SIGCSE conference in Baltimore at the end of February. The neural net was trained using data from the late 90's that was gathered to classify written zip codes for mailing purposes. We were able to download the trained neural net as a Caffe model

and have just started using it we some cropped out images with good results. This will be wrapped into the Android app once finished to classify our digits.

5 Conclusion

Overall, this project has been much more difficult that initially anticipated. The learning curve Android was not expected. Along with that, the problems in classifying digits with the small toolset learned at the DIVAS boot camp was helpful, but needed to be expanded. OpenCV was the only solution, but had little documentation so extra time was needed to understand new functions. The neural network was a great relief in making progress towards the final version of the project. We plan to have a prototype application completed shortly with continued work to be done for months to come.

References

[1] Delahaye, Jean-Paul. “The Science Behidn Sudoku.” 2006, www.cs.utexas.edu/~kuipers/readings/Sudoku-sciam-06.pdf.
[2] Felgenhauer, Bertram, and Frazer Jarvis. “Enumerating Possible Sudoku Grids.” www.afjarvis.staff.shef.ac.uk/sudoku/sudoku.
[3] Meysenburg, Mark. “Image Processing.” Image Processing, mmeysenburg.github.io/image-processing/.
[4] “OpenCV Library.” OpenCV Library, docs.opencv.org/.
[5] “ Origin of Sudoku.” The Origin of the Sudoku Puzzle, Sudoku Dragon, 2005, www.sudokudragon.com/sudokuhistory.htm.