SudokuOCR

Vitalii Zakharov
University of Tartu
vitalii.zakharov@outlook.com

1. INTRODUCTION

This project based on the previous work of Artjom Lind, Rauno Ots, Vilen Looga, and Murad Kamalov at the University of Tartu, Estonia. Author's goal was to create software that can take image, taken on the mobile phone camera, with Sudoku as an input, process it and solve Sudoku, which is a content of the image. Researches of the base work had no experience working with image recognition; author of the current project is in the same situation. Another goal of researchers was to apply knowledges gained from data mining course. They figured out how to use classification algorithms for numbers recognition, tried some of them, and picked the one that performs the best for their purposes. Another task for them was to find out how many data is needed to train classification algorithms. This project gave all the developers opportunity to gain new knowledges, clear understanding of things which are hidden under the mysterious OCR, and digging dipper into data mining course in order to understand how classification works. As a result, software for solving Sudoku using received content from an image was developed. Since the previous implementation was created using opencv 2.4 that was using c++ methods as they are just in python syntax but now opencv 3.0 is released, it is tightly coupled with NumPy library and a lot of useful features were encapsulated it was decided to write the new version of the program.

2. RELATED WORK

2.1 Thresholding

Nowadays, there are no options for OCR applications to process colored image that is why all OCR applications need to convert input image to grayscale (monochrome). There are two basic methods for thresholding: global thresholding and adaptive thresholding. Global thresholding is the simplest method. It takes an image, goes through all the pixels, calculates intensity ((R+G+B)/3) and compare it to a global thresholding value (256/2 = 128). If a pixel intensity is larger than thresholding value than the pixel is converted to white, otherwise to black. Global thresholding it is not useful in real world.

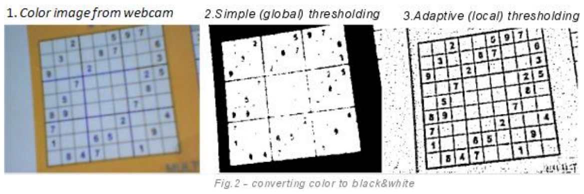

1. Color image from webcam
2.Simple (global) thresholding
3.Adaptive (local) thresholding
Fig.2 - converting color to black&white

Fig. 1 Comparison of image thresholding

Adaptive thresholding is used to get a better result. It does not use a fixed thresholding value (128), the value is calculated for each pixel separately. For instance, it takes 11*11 (121) surrounding pixels, computes the sum of their intensities and divides it by the number of pixels (121). If the mean intensity of the current pixel, located in the center of that area, is greater than the mean thresholding value, it will become white, otherwise black. This algorithm needs to do all of that for each pixel so it is much slower than the previous but it gives much better results.

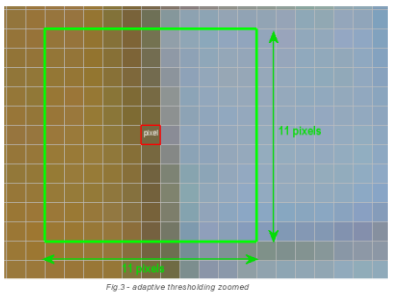


Fig.2 Finding logic of adaptive thresholding value

2.2 Contour and corner detection

Researchers decided to make an assumption that Sudoku grid on any image should have strong borders and take from 10 to 100 percent of the image.

In order to start the image processing they applied adaptive thresholding. Returned image was suitable for finding and extracting contours. Basically, numbers and lines on that image were turned to black while the background and other light pixels were turned to white.

Code responsible for that consists of contour scanner that iterates through all contours and stores threshold-passed contours to an array.

A contour is a list of points. Its representation can be different depending on the circumstances.

In order to find a contour in a binary image all connected components (pixels connected to each other) in the image should be considered. In this case, 4-connectivity of neighbor pixels is considered (see Figure 3 below), in order to find connected components.

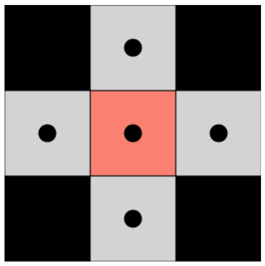


Fig. 3: 4-connectivity pixel matrix

Then it founds the largest contour and applies polynomial approximation, which means it discovers all the straight lines in the contour. For a typical Sudoku grid four lines create bounding rectangle that returns coordinates of lines intersections. If exactly four lines are found then four corners are returned and it is supposed that Sudoku grid has been detected.

$$10 < \frac{S(contour)}{S(image)} \times 100 < 100$$

Formula 1**.** Contour area threshold expression

2.3 Extraction

They took the maximal area contour, computed the bounding rectangle of this contour and extracted it. The extracted rectangle and its corners are passed to the linear transformation method.

Limitations. If an image quality is quite poor, Sudoku grid borders are broken into pieces or do not exist than it is not possible to detect grid corners and extract the Sudoku grid from the image. However if the image quality is good enough than this algorithm is very sufficient because it performs well and it is reliable enough.

2.4 Hough Transformation

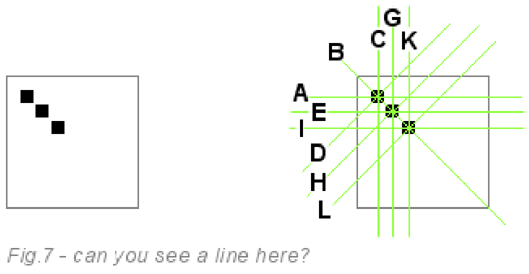


Fig. 4 Hough lines transformation

Hough transformation is a numerical method algorithm used to extract features from an image. In this case, it extracts lines. It skips all the white pixels and draws 180 virtual lines through each black pixel. Pixels ‘vote’ for the lines and the virtual line with the largest number of votes in the accumulator is the winner and most probably the real line.

Researchers used this algorithm to extract Sudoku grid corners in case if the previous extraction algorithm have not been used so it is just an option.

2.5 Transforming Image

When authors received the grid corners coordinates, they extracted them and applid linear transformation in order to get a square shape image. They used perspective transformation equations:

$$x = \frac{a \times x + b \times y + c}{g \times x + h \times y + 1}$$

$$y = \frac{d \times x + e \times y + c}{g \times x + h \times y + 1}$$

In these 2 equations there are 8 unknown variables but they could be found using construction of 8 equations system (four for each of two equations). Using this system, they were able to map points from distorted image to a flat square image.

2.6 Segmentation

This part of the program is responsible for determining where the numbers on the image are. Usually segmentation is the hardest part of the OCR process but, in this case, software should process standard 9 by 9 Sudoku, which grid is square and its size is already known, that is why there is no need to do complex segmentation operations. Authors just computed cell size and cut the image into 81 equal parts.
After that they extracted small area in the middle of a rectangle, checked whether something is in it, if not than it meant that no number was in the rectangle. Otherwise, they expanded selected area until it has covered all the number (all surrounding pixels are white). Extracted number was sent to the OCR.

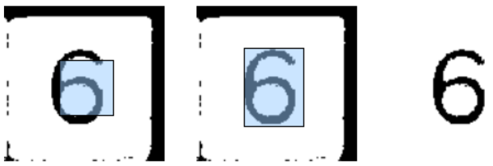


Figure 5. Extracting a number from the segment.

2.7 Optical Character Recognition

The purpose of the OCR is to take an image, process it and return character that is presented on the image.
Every recognition algorithms has steps:

1. Determine input image features
2. Train a classification algorithm with some training data.
3. Classify input image

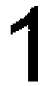


Fig. 6 Example of an input image

Determine input image features.
Figure 6 represents an input image for the OCR algorithm. Images may have different parameters (shape, color scheme, borders, etc.). Before classifying, it is needed to bring all the images to the same standard (shape, binary color). For this software, researchers resized images to 40 by 40 and vectorized image matrix because vector must represent an image in a classification algorithm. Optimal size of the image was chosen by experiment, what was found out is that for sizes less than 30 there were many wrong recognitions due to the lack of information, and if the size is more than 50 performance significantly dropped. They also changed pixel values to zero for black and one for white. That was done because SVM classification algorithm requires that kind of representation.
Training the classification algorithm. Researchers collected more than 40 samples from different Sudokus for each number. They used Support Vector Machine (SVM) with linear kernel. OpenCV SVM implementation was used for classification.
Classification of Sudoku numbers. Classification method of authors does next, it receives matrix with segments matrices in positions where some numbers are supposed to be and NONE objects where it is not. Than it transposes matrix to a vector and passes it to the classification algorithm. SVM returns the class of the classified image, which is the needed number. Their method returns matrix with integer numbers or dashes (there was no number).

2.7 Solving Sudoku

Researchers used solution of Peter Norvig. There are two functions: constraint propagation and search. This functions are used together in order to solve every Sudoku that has an existing solution. Algorithm propagates values received from the OCR to the neighboring cells. This step is called constraint propagation (CP) and for most of Sudokus it is enough to solve them. If there were more than one competitor for the same cell then the search is applied, it tries all the competitors one-by-one in a brute-force manner. Algorithm could use constraint propagation again in order to speed up. As a result, solving every Sudoku takes a few milliseconds on an average PC.

3. MY APPROACH

The goal of the project was to rewrite to the newer version of the OpenCV and improve software of Artjom and others.

3.1 Extraction

This part was partially modified. Basically contour detection was left as it is but logic of the usage was changed. If contours have not been found for thresholding bounds program starts increasing upper bound and decreasing lower bound until any sufficient contour is found. If they exist than maximal contour is taken, it is supposed to be the largest. Approximation of contour is the next step, while in the previous work they extracted bounding rectangle from the contour, I found out that standard OpenCV methods usually return not very appropriate bounding rectangles even though any contour is detected correctly. I checked approximation method and found out that it also returned points, which were lines intersections, and in that case, they were corners of the grid. These points were perfectly identified and pointed straight to where they should be.

3.2 Transformation

In this section code was pretty much changed while the meaning is left as it is. Perspective transformation is applied and perspective matrix is constructed using corner points from the extraction part. I computed width and height of the new image and using this shape and corner points OpenCV constructs perspective matrix.

3.3 Flooding

It was completely changed. In the previous version, they were looking for all the small connected components and just coloring them into white (background color of thresholded image). In my case, I used the OpenCV method fastNlMeansDenoising which removes noise from the image. I practically found parameters, which gave the best results and good performance. Image after this method was partially blurred so I changed all the pixels that are less than 128 to 0 and others to 255. On this stage image was also inverted which means that we have black background and white numbers and grid lines.

3.4 Segmentation

This section was also modified pretty much. I cut the image into 81 equal pieces, the same was done in the previous work. Then I went through all the pieces and check whether they contained a number or not. I did that as the previous researches but I selected area, which was half of the size of the piece and was centered. If the sum of the pixels inside was equal to 0 it means that it was empty. Otherwise, piece of the image was sent to the clear_borders method, which picks the middle point and finds all the contours. Algorithm works as follows, it tries to find the biggest and the closest to the center point contour. Found contour is supposed to be our number. Then it expands the size of the image until it is not equal to 40. Segmented images are stored in the matrix of image arrays and Nones.

3.5 OCR

For the OCR it was decided to use KNearest neighbor classification algorithm instead of SVM. It was much easier to set it up, while the results are the same. The logic beneath is quite the same, it is trained with a set of vectorized data as in the previous work but the logic for classification is different. The output of the algorithm is class membership. Neighbors are voting and the one who earns the majority wins, which means that input will gain that class which is the most common among its neighbors.

4. RESULTS

I used the same samples as the previous researchers. There were two phones used: Nokia N95 with 5MP camera and Nokia 5130 with 2MP camera. They made pictures of Sudoku grids from different perspective and at different angles.

In my case, for N95, program recognized 19 images out of 26. Two of them were solved incorrectly.

Here is the program workflow:

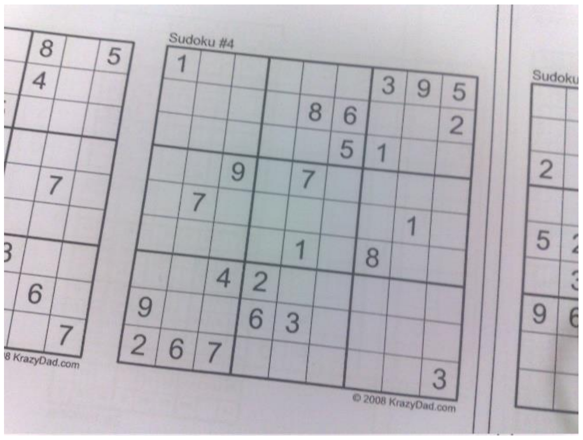


Fig. 7 Initial image

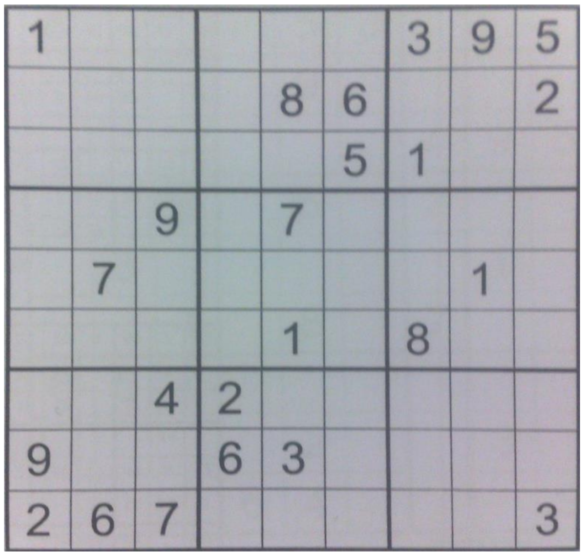


Fig. 8 Extracted image

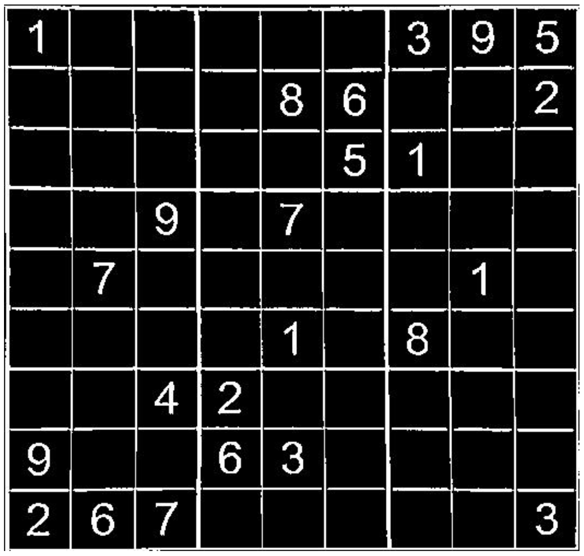


Fig.9 Flooded image

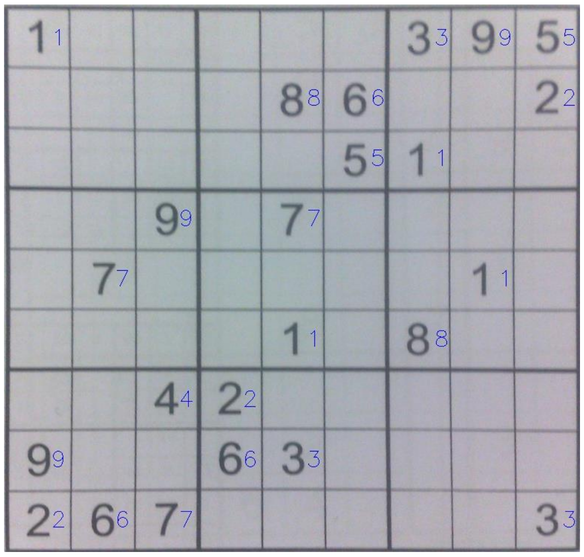
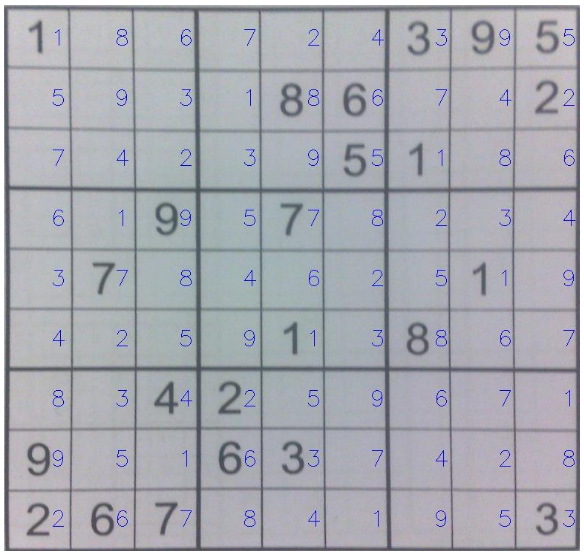


Fig. 10 Recognized numbers



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

Fig. 11 Solved Sudoku

Faulty recognition accidents occur because of not linear background color, it makes thresholding pretty challenging. In addition, quality of the camera optics makes a great influence on the image while the number of pixels is not so important.

5. FUTURE WORK

Basically, there are a lot of ideas how the program could be improved. The first and the most important task is to improve recognition under the low lightning conditions. The next task is to improve grid detection when the grid borders are thick or dashed, for instance. And the last but not the least is to make universal Sudoku solver, which means that program should solve not only 9 by 9 Sudoku puzzles. Many lines of the code should be changed to make all of these possible.

6. CONCLUSION

To sum up, I have developed a software for reading Sudoku puzzles from an image and solving them. If the image quality is quite good, made by a modern smartphone, program produces good results. After some improvements it could performs much better and recognize almost all grids. In case of a bad quality, recognition failures occur more frequently, if the image is noisy, has many gray colored components in the background or made under poor lightning conditions.

7. ACKNOWLEDGMENT

I would like to express my special thanks of gratitude to my supervisor Artjom Lind as well as the head of Distributed Systems research group professor Eero Vainikko who gave me good opportunity to do this wonderful project – Sudoku solver, which also helped me to learn about so many new things.

8. REFFERENCES

1. SudokuOCR - Project Report (MTAT.03.183 Data Mining). Artjom Lind, Rauno Ots, Vilen Looga, Murad Kamalov
2. Realtime Webcam Sudoku Solver (http://www.codeproject.com/Articles/238114/Realtime-Webcam-Sudoku-Solver)
3. Contours - 1 : Getting Started (http://opencvpython.blogspot.com.ee/2012/06/hi-this-article-is-tutorial-which-try.html)
4. Contours - 3 : Extraction (http://opencvpython.blogspot.com.ee/2012/06/contours-3-extraction.html)
5. Thresholding (http://opencvpython.blogspot.com.ee/2013/05/thresholding.html)
6. Sudoku recognizer (http://www.shogun-toolbox.org/static/notebook/current/Sudoku_recognizer.html)
7. 2008 O'Reilly Media "Learning OpenCV - Computer Vision With OpenCV" Gary Bradski, Adrian Kaehler