COMP 4102 Project:

Sudoku Solver

Team: *AMB*

Course Code: COMP4102A

Instructor: Rosa Azami

Amro Elsisy - 100917564

Menna Mohsen - 101007934

Sanjida Sanwar - 100964381

1. *Abstract:*

Sudoku grab is a collection of image processing methods that are used to produce an interactive sudoku game for users to interact with. A standard Sudoku puzzle consists of a square grid and numbers from 1-9. Each square grid consists of 9 other grids. The goal of our team in this project is to solve Sudoku puzzles using image processing techniques and a sudoku solver algorithm and implement it on pictures captured from sudoku puzzles printed on paper.

2. Introduction:

The language of implementation that our team has chosen is python. We have chosen the Sudoku solver in particular to implement because it integrates different and various elements of computer vision in it. This is done when a user takes a picture of a puzzle that is printed on paper. The application then transforms this picture into a digital solvable puzzle.

2.1 Steps of the process

There are four major steps that are involved in the process of transforming and solving the puzzle. The first step is the pre-procession of the image where we are able to work with a digitally formatted version that got executed after a picture has been taken using a cellphone camera. After the image has been pre-processed, drawing the grid comes next. This is where we implemented image processing and thresholding techniques to try to detect the puzzle's edges. Next, comes the optical character recognition and that works by extracting and recognising the numbers that are present in the puzzle. Finally, implementing the Sudoku solver algorithm to solve the puzzle.

The 4 major steps summarized:

- Image Processing
- Drawing of the grid
- Optical Character Recognition (OCR) using Pytesseract
- Applying the algorithm and solving the puzzle

2.2 The challenge:

A few challenges have faced our group while we were working on our project. The most challenging part of the implementation process was recognising the characters in the puzzle using the Optical Character Recognition (the OCR). Getting the system to recognise the grid and draw it went a lot smoother than getting character recognition to function.

3. Background:

Major image processing and computer vision concepts are integrated within the Sudoku Solver. OpenCV is a library that includes many functions that help with processing of images. Through using the help of OpenCV library and the use of their functions, the process of implementation is eased out.
Some of the few sources that we have found were useful, they included previously implemented projects that involved image processing of the sudoku puzzle.[1][2][3][4]

4. Approach

We start the process by taking a snapshot of a puzzle from paper. There are several processes that take place in order for the image to be transferred successfully digitally for the computer to be able to see it.

Prior to processing the image, the picture of the Sudoku puzzle grid gets taken by a phone camera. That causes the image to have some inconsistencies. These inconsistencies could include not being smooth or properly cropped. Prior to processing the image, a few operations have been applied to the image.

Below is an example of an image before any application of any processing techniques:

4.1 Pre-processing operations that have been applied

1. Image gets resized first[5]
2. Using the Bilateral Filter *cv2.bilateralFilter* removes noise while keeping the edges sharp using a function called pixel difference.[6]
3. *cv2.getStructuringElement* which is used to get elliptical/circular shaped kernels, *cv2.morphologyEx* and *cv2.normalize* help with the removal of noise from the image[7]
4. Erosion to close the small holes in the object which is part of the morphological operation[7].
5. Thresholding the image using cv.adaptiveThreshold which changes the image to black and white.[8][9]
6. Inverting the colour of the image using cv.bitwise_not.[10]
7. Dilating the image to increase the thickness of lines to be more visible[7]

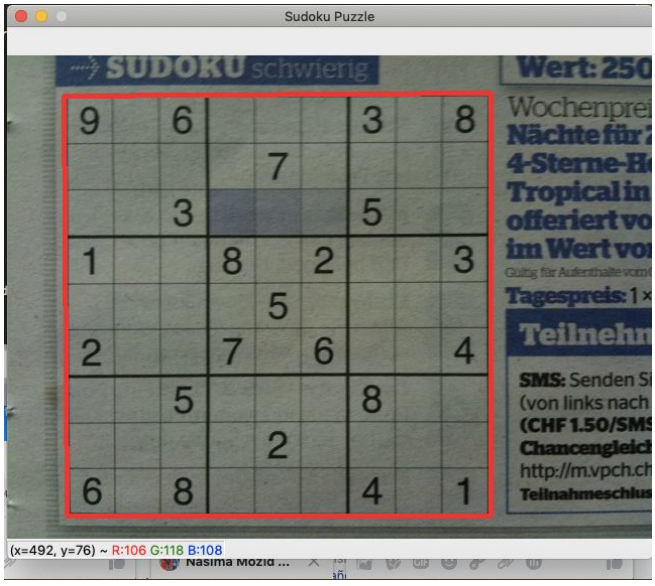
The captured image gets **resized** using the resizeSudokopuzzle.py. This ensures that we work with consistent values throughout the whole process.

After pre-processing techniques and functions have been applied, the image will look like this:

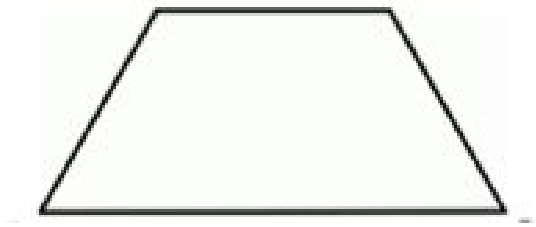
4.2 Processing the image

After preprocessing the image, the concern becomes edge detection. Applying a few more operations to the image include contouring and cropping. Finding contours helps with edge detection and corner detection. After the corners have been detected, cropping can be more easily done.

Contouring the image using *cv.findcontours*:[11]



After the application of the contouring function, we are now concerned about the cropping of the image. We need to crop the image since the human hand will likely not take a perfect photo. This will lead the sides of the image to not look equal. For example, taking a picture from a tilted angle could lead the square picture to look something like this (this is based on the camera perspective of the photo):



For that reason, a few cropping and image unification techniques need to be applied to achieve a more uniform looking image.

We do so by performing a few calculations. We calculate the longest distance between the points and get the maximum distance. This will give us the longest edge. We do so by using the OpenCV function *cv2.contourArea*.[11] This calculation will help us decide the dimensions of the puzzle or the grid that we have at hand. (Note: since we are dealing with a square grid, the longest distance will be the height and the width of the grid)

Now, the cropping can be performed. We are cropping the image using two OpenCV functions: *getPerspectiveTransform* and *warpPerspective* which will allow us to get a bird's eye view of the puzzle.[12][13]

After the application of both functions, the image looks like this:

This way the puzzle occupies the image and is easier to work with.

The next step in the image processing process is finding the largest contour using the *.findLargestContour* function as well as the *FindlargestFeature* because in some cases one does not work as perfectly as the other and vice versa. In both we used cases we used the cv function *cv.findContour.* [1][14][15]

How do we decide which one to use? By getting a ratio of the *largestfeatureArea/largestcontourArea*. We can't use just either largest feature or largest contour because in some cases we can't expect to find the grid using just largest feature, we need the largest contour as well. If the ratio is between 0.95 and 1.5 then we use *largestfeatureArea* otherwise we use largest contour. This will result in 81 cells to be produced.

We needed some helper functions that were written. *FindlargestFeature* function can be found in *display* class in *helper.py*. This function finds the largest connected pixel structure in the image and returns the seed of it.

The methodology that we used to find the largest feature using the function *FindlargestFeature* in image was similar to the methodology used to find largest contour in the function *findLargestContour.* The difference here is that we used *cv2.floodFill* which fills the bounded area (cell) with the chosen color, grey, using the function *cv2.floodFill*.[16]

Another helper function that was used was the *computeBoundingBoxofFeature* and it was used to flood all the images with grey, then it flood fills all the zero value pixels with white. All the grey pixels with black. Looping through all the pixels, we can then examine the whites and compute the corner points.[4]

Finally, the sudoku solver class is located in the *solve_sudoku_from_image.py*, it finds contours and then sends those contours to the *storeDetectedDigits* function. We then use a for loop to iterate over the contours, we get the boxes parameters using *cv2.boundingRect* function which in return helps us get the rectangle size.[3]
x,y,w,h = cv2.boundingRect(cnt)
*rect_size = w*h*

We will then need the coordinates of the centre of the cell followed by the coordinates of the actual cell. This is when we can crop the part where a digit has been found. To find the *detected_digit = pytesseract.image_to_string(blurred, lang="eng", config=config)* where blurred is the extracted digit.[17] Finally, we would need to fill the empty spaces in the Sudoku puzzle with 0s using the *fillEmptySpaces* function. The digits now have been extracted and sent to the pytesseract engine. We then store it in the Sudoku array using *self.sudoku[self.rows[y_coord]+self.columns[x_coord]] = str(detected_digit).*[4]

The sudoku string from pytesseract is then passed into the solving algorithm class *sudoku_solver.py* and after getting the results, we would then need to print them on the cropped image of the puzzle, we do that by using the *printSolution* function that uses cv2.putText that

allows us to print on an image with the font and colour of choice.[18] Note that: The picture is resized, once to extract the puzzle, and then resize the image again to rescale intensity to get the ocr engine to read the digits.

4.3 Dependencies:

- Numpy
- Tesseract-eng
- Scikit-image
- Imutils

4.4 Command to Run:

The command to run the software is: ./sudokuImageSolver.py

5. Results:

The solution gets printed using *printSolution.* We will use two functions from the *sudoku_solver* file. They are: *generate_string_from_sudoku* and *solve_sudoku.*

The font used for the solution is *font = cv2.FONT_HERSHEY_COMPLEX*

We will use the function *cv2.putText* to put in the text on the image with the respective positions *x* and *y.*[16]

After all the previous operations have been applied, the resulting image with the printed characters will look like this:

6. Solving Algorithm:

Sudokus are formatted as a string where the rows are concatenated and empty spaces are represented as “0” or as “.”.[3]

"100200040020003900907000500004000057000541000350000100003000709001400080090002006"

Step 1:

Dictionary with the cell name as a key and name of row, col, block in this cell as the value in *comb_dict*

Step 2:

Create a dictionary from a sudoku string using *create_dict_from_string(grid)*

Step3:

Create a dictionary from sudoku string with the cellname as key and list of possible numbers as the values in the variable *create_dict_from_sudoku_string(grid)*[3].

Step 4:

For each number in a cell, recursively call a solve function. Find possible solutions in the dictionary of possible numbers, and move numbers accordingly using this function: *recursive_solve(create_dict_from_sudoku_string(sudoku_string))*[3]

7. List of Work:

- Equal work has been performed by all participants on the group

8. GitHub Page:

username: COMP4102AMBlink to page: https://github.com/AmroElsisy/COMP4102AMB

9. References:

[1] Patel, N. (2017, November 6). Solving Sudoku: Part II. Retrieved April 21, 2020, from https://medium.com/@neshpatel/solving-sudoku-part-ii-9a7019d196a2

[2] How does it all work? (2009, July 19). Retrieved April 21, 2020, from http://sudokugrab.blogspot.com/2009/07/how-does-it-all-work.html

[3] KleinSamuel. (n.d.). KleinSamuel/sudoku-solver. Retrieved April 21, 2020, from https://github.com/KleinSamuel/sudoku-solver

[4] Foo, C. (n.d.). fzy1995/SudokuImageSolver. Retrieved April 21, 2020, from https://github.com/fzy1995/SudokuImageSolver

[5] Resize an image without distortion OpenCV. (2017, June 20). Retrieved April 21, 2020, from https://stackoverflow.com/questions/44650888/resize-an-image-without-distortion-opencv

[6] Smoothing Images. (2019, December 31). Retrieved April 21, 2020, from https://docs.opencv.org/master/d4/d13/tutorial_py_filtering.html

[7] Morphological Transformations. (n.d.). Retrieved April 21, 2020, from https://docs.opencv.org/trunk/d9/d61/tutorial_py_morphological_ops.html

[8] Image Thresholding. (2019, December 31). Retrieved April 21, 2020, from https://docs.opencv.org/3.4/d7/d4d/tutorial_py_thresholding.html

[9] Thresholding (image processing). (n.d.). Retrieved April 21, 2020, from https://en.wikipedia.org/wiki/Thresholding_(image_processing)

[10] Arithmetic Operations on Images. (n.d.). Retrieved April 21, 2020, from https://docs.opencv.org/2.4/modules/core/doc/operations_on_arrays.html

[11] Structural Analysis and Shape Descriptors. (2019, December 31). Retrieved April 21, 2020, from https://docs.opencv.org/2.4/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html

[12] Rosebrock, A. (2014, August 25). 4 Point OpenCV getPerspectiveTransform Example. Retrieved April 21, 2020, from https://www.pyimagesearch.com/2014/08/25/4-point-opencv-getperspective-transform-example/

[13] Rosebrock, A. (2014, May 5). Python and OpenCV Example: Warp Perspective and Transform. Retrieved April 21, 2020, from https://www.pyimagesearch.com/2014/05/05/building-pokedex-python-opencv-perspective-warping-step-5-6/

[14] Cartucho, J. (2017, June 16). Find and draw the largest contour in opencv on a specific color (Python). Retrieved April 21, 2020, from https://stackoverflow.com/questions/44588279/find-and-draw-the-largest-contour-in-opencv-on-a-specific-color-python

[15]Contour Approximation Method. (2019, December 31). Retrieved April 21, 2020, from https://docs.opencv.org/trunk/d4/d73/tutorial_py_contours_begin.html

[16] Mallick, S. (2015, November 23). Filling holes in an image using OpenCV (Python / C). Retrieved April 21, 2020, from https://www.learnopencv.com/filling-holes-in-an-image-using-opencv-python-c/

[17] pytesseract 0.3.4. (n.d.). Retrieved April 21, 2020, from https://pypi.org/project/pytesseract/

[18] Ranjis09 Python OpenCV: cv2.putText() method. (2019, August 28). Retrieved from https://www.geeksforgeeks.org/python-opencv-cv2-puttext-method/

Other useful references that helped in giving us an idea of the whole image processing methodologies:

Qi Wan, R. (2019, December 31). Vision-Based AI Model Solves Sudoku at a Glance.Retrieved April 21, 2020, from https://syncedreview.com/2020/01/01/vision-based-ai-model-solves-sudoku-at-a-glance/

Connected-component labeling. (n.d.). Retrieved April 21, 2020, from https://en.wikipedia.org/wiki/Connected-component_labeling

Hough transform. (n.d.). Retrieved April 21, 2020, from https://en.wikipedia.org/wiki/Hough_transform

Optical character recognition. (n.d.). Retrieved April 21, 2020, from https://en.wikipedia.org/wiki/Optical_character_recognition

Pattern recognition. (n.d.). Retrieved April 21, 2020, from https://en.wikipedia.org/wiki/Pattern_recognition

Projects. (n.d.). Retrieved April 21, 2020, from https://opensource.google/projects/tesseract

Glossary of Sudoku. (n.d.). Retrieved April 21, 2020, from https://en.wikipedia.org/wiki/Glossary_of_Sudoku#Sudoku_variants
Killer sudoku. (n.d.). Retrieved April 21, 2020, from https://en.wikipedia.org/wiki/Killer_sudoku

Python OpenCV: cv2.putText() method. (n.d.). Retrieved April 21, 2020, from https://www.geeksforgeeks.org/python-opencv-cv2-puttext-method/