APPM4058A & COMS7238A Project – Report

Abdulkadir Dere - 752817, Jared Tremayne Naidoo - 719238

15 June 2020

1 Introduction

This project aims to detect the numbers within a given Sudoku puzzle using strictly image processing techniques and produce a 2D output array to be consumed by a Sudoku puzzle solver. The output should be written to a text file which can be consumed by the Sudoku solver library.

2 Problem Domain

The problem domain for this project is to apply image processing techniques on a given Sudoku puzzle image. The image processing techniques span the following domains: image thresholding, connected component analysis, boundary detection and the filtering of images using mathematical morphological operations.

3 Dataset

The data set used to evaluate the performance of the algorithm consists of 14 different Sudoku puzzles. The Sudoku puzzles were obtained from the website Sudoku Puzzles 99 [1]. Each puzzle was obtained from the same source however to add complexity to the problem variations in images size were used. Examples are provided below for two puzzles with varying size (Figure:1). These images are in RGB format.

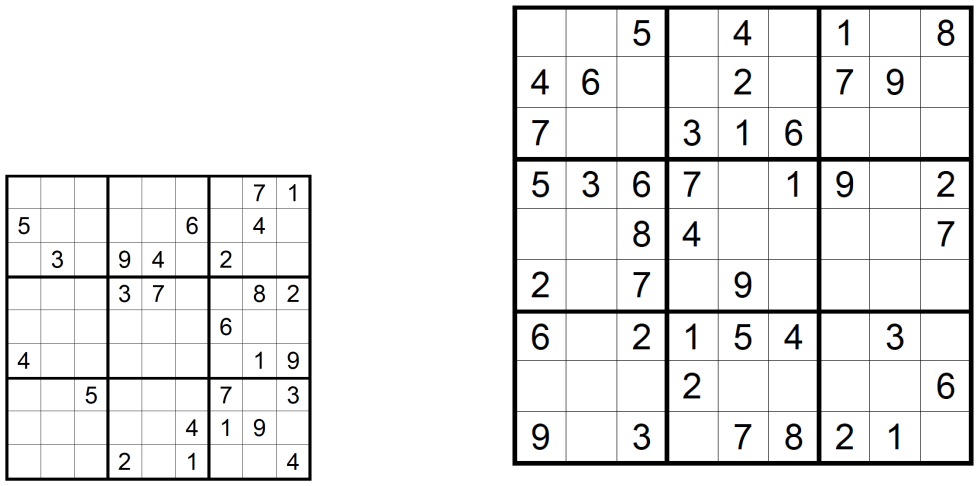

Figure 1: Sudoku puzzles with varying sizes

4 Methodology

This section discusses image processing methods and techniques applied to the Sudoku puzzle to extract the digits 1-9 of the puzzle. This program was developed using MATLAB.

4.1 Original Image and Pre-processing

The RGB image received from the source is converted from a colour image into a grayscale image. This conversion aims to prepare the image for the application of a thresholding algorithm which will convert the image into a binary format (Listing:1). The complement of the binarized image is taken so that the Sudoku puzzle is identified by the foreground pixels.

```
% Convert to grey scale image
grayImage = rgb2gray(sudoku_image);
% Convert to a binary image
binary_image = imbinarize(grayImage);

% Take complement of image, format suitable for BW label. Foreground
    pixels must be white
comp_image = imcomplement(binary_image);
```

Listing 1: Image pre-processing

4.2 Identify and Extract the Sudoku Frame

Once a binary image is obtained we move on to identifying the outer frame and the lines that interconnect the different blocks of the Sudoku puzzle (Listing:2). This is achieved through the use of Connected Components. The aim here is to find the largest connected component of the image. The largest connected component would be the Sudoku puzzle's frame and the interconnected lines of the frame (Figure:2). The image is cropped to the width and height of the frame so any unnecessary pixels outside of the frame is removed from the image.

```
% Extract the connected components, specifically the largest connected
% component. This is the frame and the interconnected horizontal and
% vertical lines
sudoku_frame = bwareafilt(comp_image, 1, 'largest');

% Crop the frame, remove any white space outside of the frame of the
% puzzle
[row, column]=find(sudoku_frame==1);
cropped = sudoku_frame(min(row):max(row), min(column):max(column));
```

Listing 2: Extracting frame using connected component algorithm

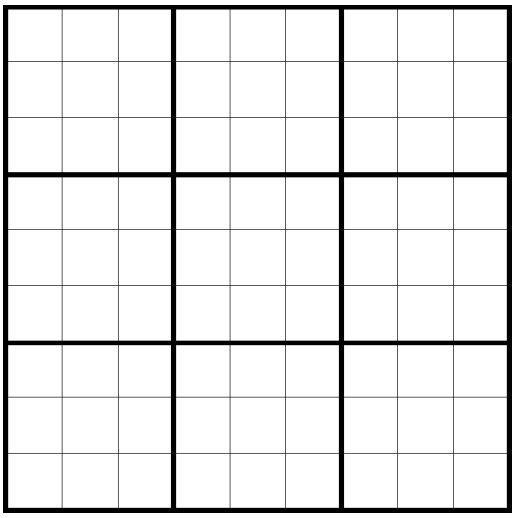

Figure 2: Largest connected component (Sudoku puzzle's frame and interconnected lines of the frame)

4.3 Identify Digit Block and Digit Within Each Block

The cropped image allows us to identify boxes within the Sudoku puzzle image. These boxes contain digits within the range of 1-9 or are empty. The connected component process is used to identify each box as a connected component (Listing:3). Each box is separated by interconnected lines of the frame, hence each box is a connected component. This will result in 81 connected components since the Sudoku puzzle's dimensions are 9 by 9. We keep track of the coordinates (row and col) of each box's first pixel (top leftmost pixel) as well as the width and height of each box. This will help us to create a region of interest (ROI) for each box to analyse the pixels within the box.

```
    % Perform Connected Component search again on the frame this time. This
    % will give us the number of boxes within the frame.
    % Region props allows us to detect the co-ordinates of each connected
    % component. i.e. Location of each box in the sudoku frame
    stats = regionprops(imcomplement(cropped),'BoundingBox');
    % Store all the x & y coord of bounding boxes, store in a list
    bounding_box = cat(1, stats.BoundingBox);
    % Crop binarised image with numbers in it. We will work on this going
    % forward image.
    image = comp_image(min(row):max(row), min(column):max(column));
```

Listing 3: Identify each digit block using connected components

4.4 Structuring Element

The structuring element (SE) for each digit has been created and saved as a ".mat" file (Figure:3). The SEs are created by manually extracting each of the 9 digits from a Sudoku puzzle. A SE contains all of the pixel values that form a digit as well as additional ghost pixels. An example of the structuring element for the digit 9 is provided below.

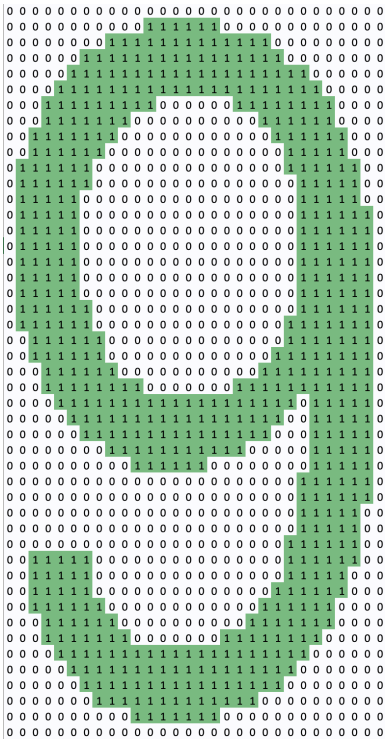

Figure 3: Structuring element for the digit 9.

4.5 Hit-or-Miss

A hit-or-miss operation is applied to each of the 81 boxes using the 9 SEs. When the pixel values within a block match perfectly with one of the 9 SEs, the digit for which that SE represents is updated in a 2D results array at the corresponding location (Table:1 and 2). The hit-or-miss process scans each pixel in a box until it gets to a pixel where it will match exactly with the given SE (i.e. The set of pixels in the SE fits in the foreground pixels of the specific block) (Listing:4). This process completes after iterating through all the SEs. All digits are identified and their locations are recorded in the results array. We skip these blocks to optimize the process as those boxes already have a valid value assigned to them. The value zero indicates empty blocks in the input Sudoku puzzle.

1	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0

Table 1: 2D results array after detecting location for the digit number 1.

1	0	2	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	2	0
2	0	0	0	0	0	0	0	0
0	0	0	0	0	1	2	0	0
0	0	0	0	0	2	0	0	1
0	0	0	2	0	0	0	0	0
0	2	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0

Table 2: 2D results array after detecting locations for the digits number 1 and number 2.

```
function result = OCR(image, bounding_box)
    % Get number of boxes
    num_box = size(bounding_box,1);
    % Create the resulting matrix. Tracking array that we will display in
    % the end
    result = zeros(9,9);

    % Import a predefined Kernel for each number from 1 to 9
    % We have created these Kernels manually
    kernels = ["num1.mat","num2.mat","num3.mat","num4.mat","num5.mat","num6
        .mat","num7.mat","num8.mat","num9.mat"];

    % Apply a hit or miss to each of the blocks using every Kernel
    for i = 1:size(kernels,2)
        kernel = load("kernel/"+kernels(i)).SE;
        kernel_value = i;
        result = hitormiss(image, bounding_box, result, kernel,
            kernel_value, num_box);
        % Display the Sudoku Puzzle
        disp(result);
    end
```

```
end

% Iterate over all the boxes in the puzzle
for i=1:num_box
    % Only check the boxes which hold the value zero.
    % Optimisation technique so we skip any box that has already
    % matched with a kernel
    if (box_value(i) == 0)
        % Specify the current box or ROI
        roi = [bounding_box(i,1), bounding_box(i,2), bounding_box(i,3),
           bounding_box(i,4)];
        % Extract it from the original image
        I2 = imcrop(image,roi);
        % Convert to int
        sub_img = uint8(I2);
        % Apply hit or miss to extracted int image
        hitmiss = bwhitmiss(sub_img, SE, SE_i);
        % Check for match
        match = check_match(hitmiss);
        % If true
        if match == 1
            box_value(i) = kernel_value;
        end
    end
end
result = box_value;
end
```

Listing 4: OCR function applies hit-or-miss transform to the 81 blocks.

5 Image Processing Algorithms

- **Connected Components**: Used to identify the puzzle's frame and each digit box within the frame. The connected component algorithm used in the context of this project allows us to easily identify the frame within the puzzle. This extraction is crucial since the frame encompasses all of the data required for the proposed methodology in the Methodology section. The connected component technique was used again to identify the digit boxes. The complement of the cropped image was taken so that the frame is represented by the background and digit boxes are represented by the foreground pixels. Each digit box is a connected component on its own. These digit boxes are enclosed and separated by the frame's interconnected lines.

- **Hit-or-miss Transform**: Used to find shapes in images [2]. Hence, we defined the shape of each digit as a SE. We applied a hit-or-miss transform, using each SE, to each block to identify which digit the block holds. This is verified as hit-or-miss and will keep the pixels which match with a SE.

6 Results

A heat-map is shown for the identified digits so results can be compared with the original Sudoku puzzle image (Figure:4). The results are written into a CSV and XLS files so the results can be consumed by a Sudoku solver program. The value zero indicates that the box is empty and needs to be computed by the Sudoku solver program.

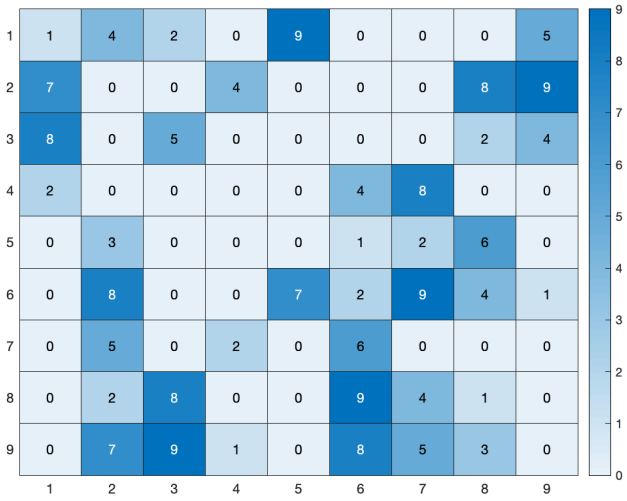

Figure 4: Heat-map result after iterating through all SEs

7 Conclusion

In this report, we looked at the extraction of digits from a Sudoku puzzle using image processing techniques. We were able to achieve this using two main techniques namely connected components and hit-or-miss transform. The solution provided has been tested across 14 different Sudoku puzzles with a success rate of 100%. The output produced has met the requirements for being in a suitable format for the Sudoku solver program.

Potential improvements could be to implement an algorithm that can create structuring elements automatically. Introducing different SEs for each digit would improve the solution's ability to extract digits from different puzzles.

References

[1] "Website Name: printable sudoku 99." https://www.printablesudoku99.com/, 2020. Accessed: 2020-06-10.

[2] R. C. Gonzalez and R. E. Woods, *Digital Image Processing (3rd Edition)*. USA: Prentice-Hall, Inc., 2006.