WITS UNIVERSITY

School of Computer Science and Applied Mathematics
UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG

APPM-4058: Digital Image Processing

Recognising Sudoku Grids in Images

Michael Beukman

June 25, 2021

1 Abstract

We develop a method of segmenting an image of a sudoku grid to crop and isolate each cell, determine which cells contain digits and finally predict the value of the digit at each position. This information can then be used to solve the given sudoku puzzle.

2 Introduction

Sudoku is a popular game that involves completing an initially seeded 9×9 grid with digits from 1 - 9 such that no row, column or 3×3 sub-block contains the same number more than once (Felgenhauer and Jarvis, 2006).

This project involves reading in images of sudoku grids, and using image processing techniques to segment and separate the digits, as well as actually recognising them. The end product is a text file that can easily be read by any sudoku solver program, and solved.

3 Method

We have two main components in our method, segmentation and recognition. Segmentation is performed first, which extracts the gridlines, separates the digits from each other, and determines where in the grid they occur.

Recognition takes the cropped digit images from the segmentation phase and attempts to classify them. Putting these two sections together, we can obtain the final sudoku grid. In the following sections we describe these in more detail.

Most of the image processing was performed using Scikit-Image (van der Walt *et al.*, 2014) and OpenCV (Bradski, 2000). We make heavy use of the OpenCV function *findContours*, which is based on Suzuki and Abe (1985). The main idea is that this algorithm finds the outer borders of different objects by finding pixels where there is a transition from background to foreground, and uses connected components to extract the entire border for an object.
We use this algorithm to find bounding boxes around digits, as well as locating the individual cells.

3.1 Segmentation

Intermediate stages of the segmentation process can be found in Figure 2.

1. We first segment the main grid, to make subsequent steps easier. To do this, we use a Sobel filter to detect the edges, threshold it using Otsu's method and remove noise using a median filter. Then we dilate once to repair broken edges and use OpenCV's *findContours* to find the largest contour. We assume this is the grid and simply crop the original image based on this bounding box.

2. We then threshold the image using an adaptive threshold, with a block size of 151, which is comparable, but slightly larger than the individual cells. This adaptive threshold was used due to unequal lighting conditions for some of the images, which might result in over- or under-thresholding when using a global threshold. We use a median filter to remove salt-and-pepper noise. This thresholded image is used in the next steps.

3. We use a combination of OpenCV's contour detection, as well as the Hough transform to determine

the locations of the small cells inside the grid by locating all of the gridlines. We perform the Hough transform to locate the prominent lines in the image, and we create a new image that contains only these lines, referred to as the *line image*. Using both of these methods is important, as we restrict our search to nearly vertical or horisontal lines using the Hough transform, but we can easily mask out the individual cells using the contours. We combine these two using $f(x,y) = d(c(x,y)) \times d(h(x,y))$, where c is the contour image, h is the line image and d is a function that performs three dilations, with rectangular 1×5, 5×1 and 5×5 structuring elements to thicken the lines slightly, and ensure there is sufficient overlap. This multiplication has the effect of performing a logical and of these two images. This step can be seen in the bottom row of Figure 2

4. We then crop the thresholded image using these identified cells, and determine which contain digits and which do not by using a median filter and determining which cells contain more than 20% white pixels. To minimise the effect of the gridlines, which are usually found on the boundary of the individual cells, we only take into account how many white pixels are in the middle ninth of the cell, and ignore the boundaries (as in the bottom row of Figure 3). To recognise the digit however, we use the whole cell.

We also perform a slightly different method depending on the image. This part is mostly applicable to cropping the image once the line image and contour image have been combined. For our method to work, this image should have 81 distinct contours corresponding to the 81 cells. We start off with a simple method, and add more complexity until this condition is met. Our first method is just as described above, but when that does not result in 81 cells, we fill in all rows and columns with 1s if those rows/columns contain at least 50% ones, with the goal of repairing broken edges. If that fails, we use $f(x,y) = d(c(x,y)) + d(h(x,y))$ instead of $\times$, to select possible gridlines more broadly using a logical or. If those do not work, we perform them again, but this time, a morphological closing is performed with vertical and horisontal structuring elements, again in a bid to fix the broken lines. If all of these different variations fail, then the method fails overall.

3.2 Recognition

To perform the recognition, we created a small training set of digits (around 175 digits), and a small testing set of about 60 digits by manually labelling some cropped digits. Using this, we attempted multiple different methods of classifications, like K-Nearest Neighbours (KNN), KMeans and Principle Component Analysis (PCA), but the best method proved to be Linear Discriminant Analysis (LDA). This method performs a similar function to PCA, and "maximizes the ratio of between-class variance to the within-class variance in any particular data set thereby guaranteeing maximal separability" (Balakrishnama and Ganapathiraju, 1998). This in effect ensures that classes are different from each other, and all the datapoints inside a specific class are similar. The model fits a Gaussian density function to the datapoints from each class. This can also perform dimensionality reduction by projecting the data to the smaller amount of discriminative directions (Pedregosa *et al.*, 2011).

The cropped grid cells were processed using the following steps before we applied LDA to them. This process was applied to both the training, testing and the final images to be predicted.

Intermediate stages for this process can be found in Figure 4

1. Threshold the cell using a global Otsu threshold, as in Figure 4b.

2. Use morphological labelling to extract the connected components, like Figure 4c.

3. Take the label with the most pixels (excluding background pixels) in the middle part of the image (again to reduce the effect of the gridlines), as shown in Figure 4d. We assume this is the digit.

4. Then simply create a bounding box around the selected label and crop the original image according to this bounding box, as in Figure 4e.

5. We then resize the image to a constant 16×16 using bi-linear interpolation (Figure 4f) and flatten it into a 256 dimensional vector.

4 Data

We use an open-source dataset[1], which was introduced in Wicht and Hennebert (2014) that contains 200 images of Sudoku grids. We removed a few badly taken ones, and used a few as testing images when developing the methods.

The images in this dataset are quite varied, both in terms of sudoku grid style, as well as lighting and image quality. We attempt to make our method work in general, but since these images are so different, this is not always possible. Figure 1 shows a few images from this dataset.

5 Results

Here we will first show some of our intermediate steps as detailed in Section 3, and finally we will discuss the performance and overall accuracy of our method.

5.1 Segmentation

In Figure 2 we can see the intermediate steps, starting with segmenting just the grid, and performing a local thresholding operation. Then we calculate the contours and Hough lines to determine where the grid lines are. Finally we combine these to crop the image into the individual cells.

5.2 Filtering

In Figure 3, we show some empty and non-empty cells. We can see that even the empty cells sometimes contain border pixels, or noise. This is why we use a threshold of 20%. Only using the middle part of the image is important, as borders can interfere with this simple detection method.

[1] https://github.com/wichtounet/sudoku_dataset

	Over All Images	Over Non Failed
Locations	78 %	93 %
Digits in Total	79 %	94 %
Perfect Prediction	56 %	68 %

Table 1: Table of final results

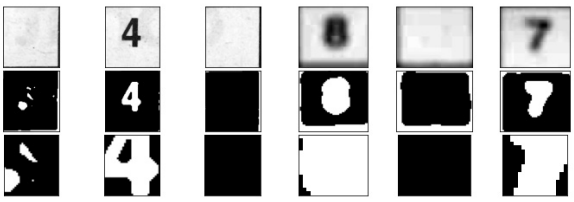


Figure 3: Empty and non-empty cells from two different grids. The first row is normal, the second row is thresholded and the bottom row is only the middle third of the image.

5.3 Recognition

Figure 4 shows the steps we took to process the cropped digits before we attempted to classify them. Figure 5 shows some predictions, and we can see it is not always accurate, potentially because the training data did not contain digits of all fonts and styles.

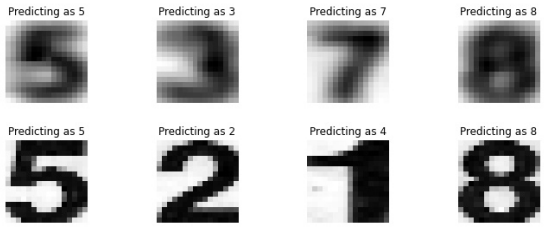


Figure 5: Some predictions

5.4 Final results

From our original 200 images, we removed 6, used 8 as testing images when developing the method and 8 as digit training data. We do include these 16 in the following results, as they were our 'training' set, and we thus only report results on unseen images.

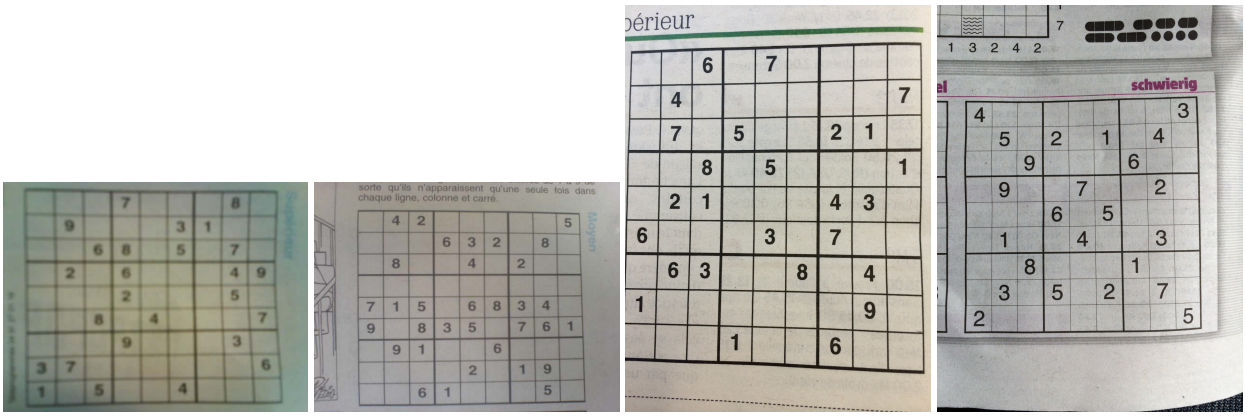


Figure 1: Original Images

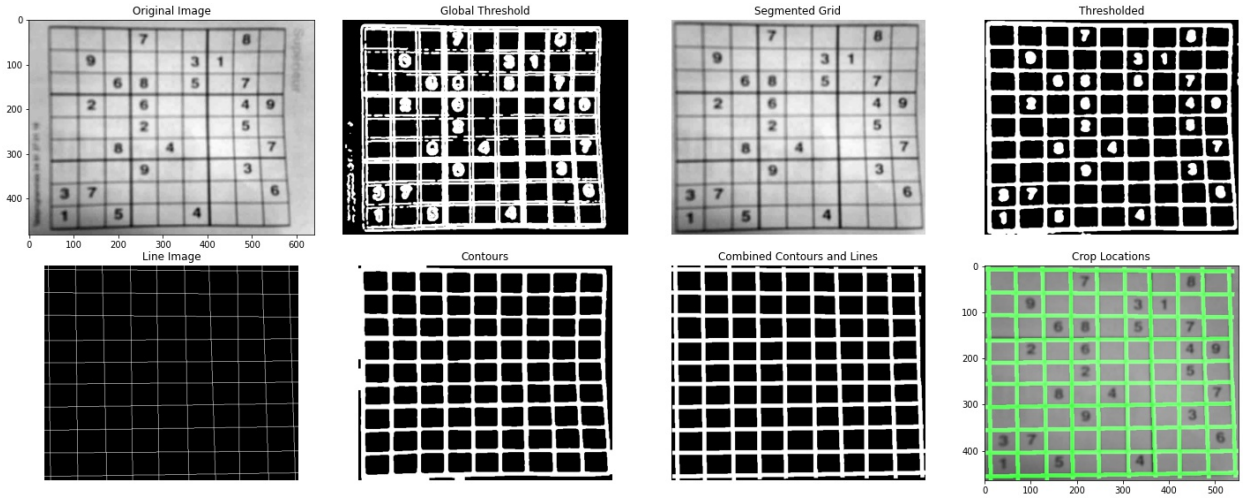


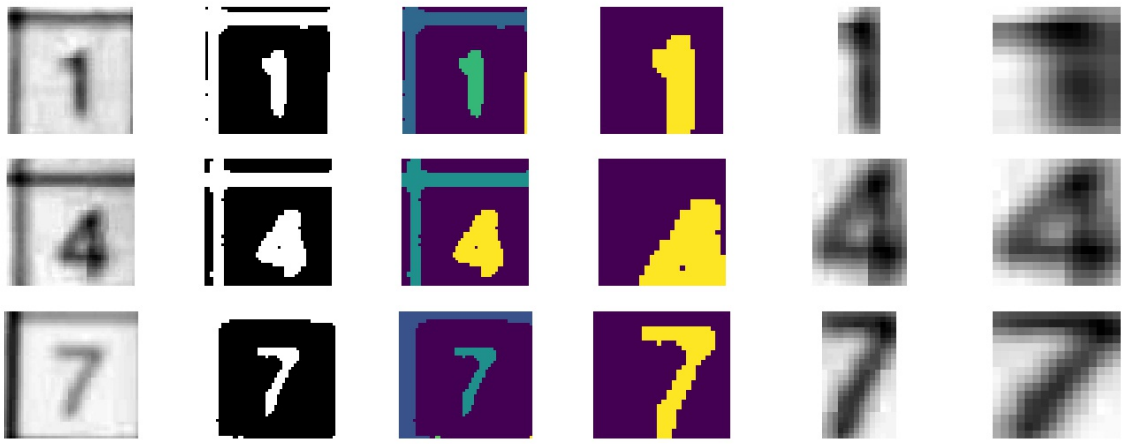
Figure 2: Intermediate Steps. This corresponds to the steps in Section 3.1.

Our results were mixed, but promising. Of a total of 178 images to test on, we failed completely with 29, because we couldn't segment the grid at all. Of the remaining 149, we managed to find the segmented grids and the location of the non-empty digits of 93%. We correctly classified all digits in 68% of these non-failed images. Out of the total number of digits that we considered, we predicted 94% of them correctly, but the incorrect predictions were spread out across multiple grids, leading to predicting at least one wrong digit in many images. We predicted the entire grid perfectly 56% of the time (which includes the times we failed to segment the grid). Table 1 details the accuracy we achieved. Figure 6 shows an input image, and the predicted grid.

5.5 Limitations and Failure Cases

While our method successfully works for a large number of images, there are still those that cause it to fail. In this section we analyse these failure cases to determine which parts of our method are not robust to different images. We developed this method by focusing on a small set of images to test our techniques on, and images largely different from these do cause problems.

In Figure 7 we show some images that failed. We can see that the lighting conditions are not very good, or there are multiple other pieces of information besides the grid in the image, or some lines in the grid are thinner than the rest.



(a) Original (b) Thresholded (c) Labelled (d) Labelled Center (e) Cropped (f) Resized

Figure 4: Preprocessing steps for digit data. This corresponds to the steps in Section 3.2

(a) Actual Image

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

(b) Predicted grid (converted from the output text file)

Figure 6: Output of our method

This in turn affects our thresholding process, resulting in a suboptimal threshold, which impacts the later stages of the pipeline. These problems are not insurmountable, and with more work, and potentially more adaptive methods, we could succeed with these failed images. The digit recognition system used was quite simple, and we only used limited training data. Increasing the amount and diversity of this training data would improve the recognition score as well.

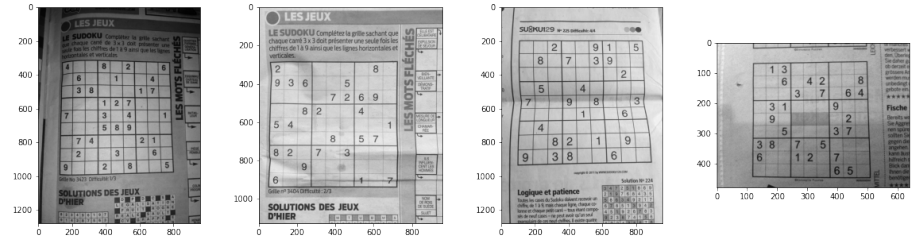
6 Conclusion

In conclusion, this report outlined a method that can segment a sudoku grid, as well as recognise the digits. It takes in an image, and outputs the resulting grid to a text file, which can easily be parsed and solved using a standard sudoku library.
Our method worked relatively well, but the method fails on new images that are substantially different from those the method was designed on.
Future work could be to address the above limitations, as well as to investigate different, more advanced digit recognition techniques. There are still improvements to be made, but the method is accurate, and relatively simple.

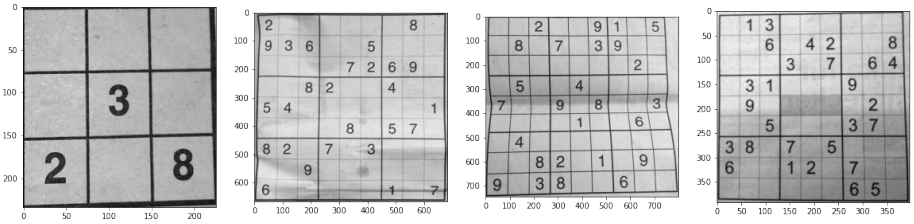
The full code can be found at https://github.com/Michael-Beukman/ImageProcessingSudoku.

References

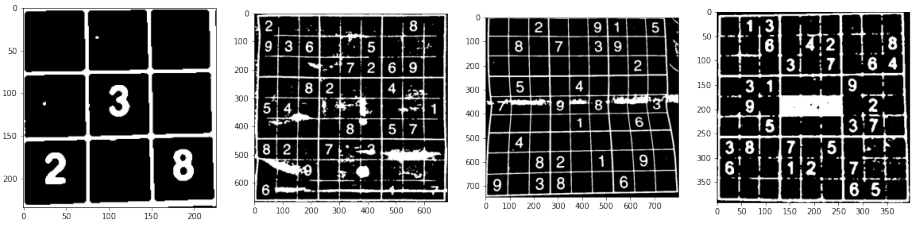
[Balakrishnama and Ganapathiraju 1998] Suresh Balakrishnama and Aravind Ganapathiraju. Linear discriminant analysis-a brief tutorial. *Institute for Signal and information Processing*, 18(1998):1–8, 1998.



(a) Original Images



(b) Segmented Grids



(c) Thresholded Grids

Figure 7: Some failure cases

[Bradski 2000] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[Felgenhauer and Jarvis 2006] Bertram Felgenhauer and Frazer Jarvis. Mathematics of sudoku i. *Mathematical Spectrum*, 39(1):15–22, 2006.

[Pedregosa *et al.* 2011] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[Suzuki and Abe 1985] Satoshi Suzuki and Keiichi Abe. Topological structural analysis of digitized binary images by border following. *Computer vision, graphics, and image processing*, 30(1):32–46, 1985.

[van der Walt *et al.* 2014] Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu, and the scikit-image contributors. scikit-image: image processing in Python. *PeerJ*, 2:e453, 6 2014.

[Wicht and Hennebert 2014] Baptiste Wicht and Jean Hennebert. Camera-based sudoku recognition with deep belief network. In *Soft Computing and Pattern Recognition (SoCPaR), 2014 6th International Conference of*, pages 83–88. IEEE, 2014.