Recognising Sudoku Grids

APPM4058 - DIP

Michael Beukman

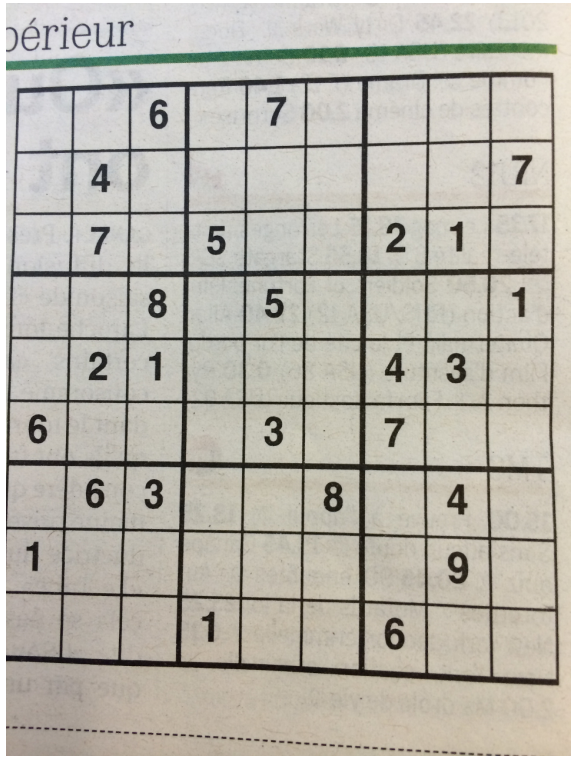
University of the Witwatersrand

June 25, 2021

Outline

Introduction

Sudoku is a puzzle that gets played by many people that involves solving a puzzle by filling in a 9x9 grid with digits from 1 - 9 with certain constraints, like no row/column/3 $\times$ 3 sub-block can contain the same digit twice.



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

Our work

Our work involved reading in an image of a sudoku grid, and using image processing techniques to (1) segment and isolate each individual grid cell, (2) determine whether each cell has a digit or not and (3) recognise the digit.

Thus our method is an end to end process that maps from images of sudoku grids to text files representing the positions and values of the initial digits.

Data

Open Source dataset[1] [Wicht and Hennebert, 2014] consisting of many different images.

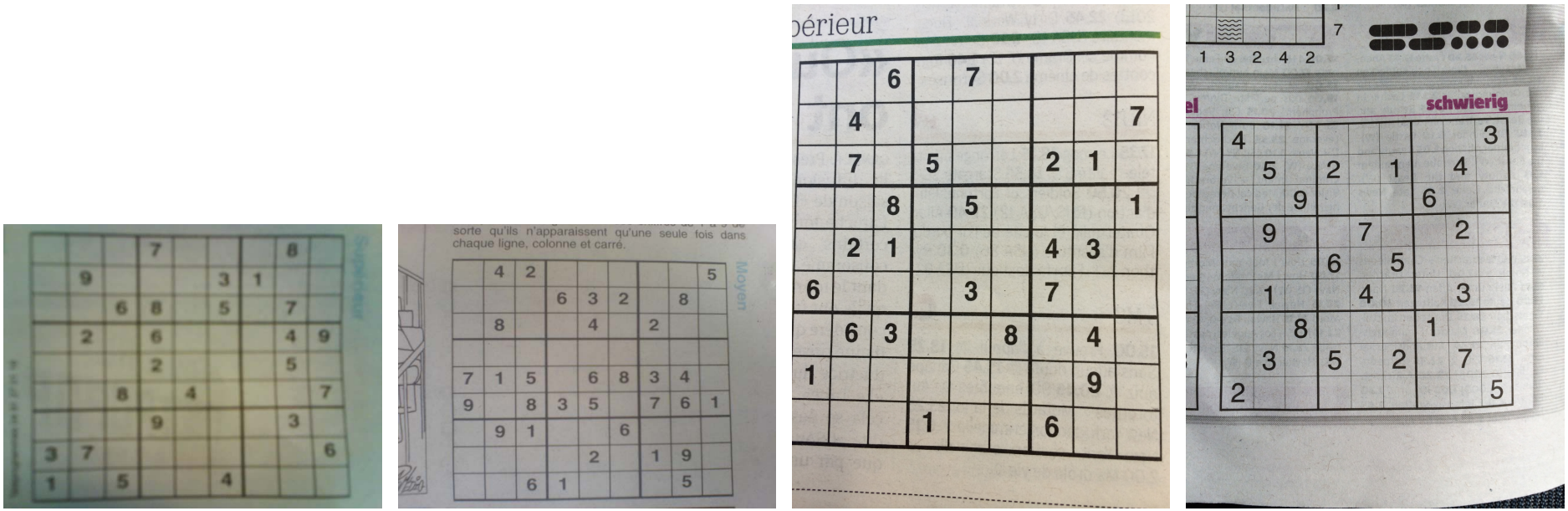


Figure: Original Images

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

Method I

First segment,

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.

Method II

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 2

Method III

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 3). To recognise the digit however, we use the whole cell.

Results

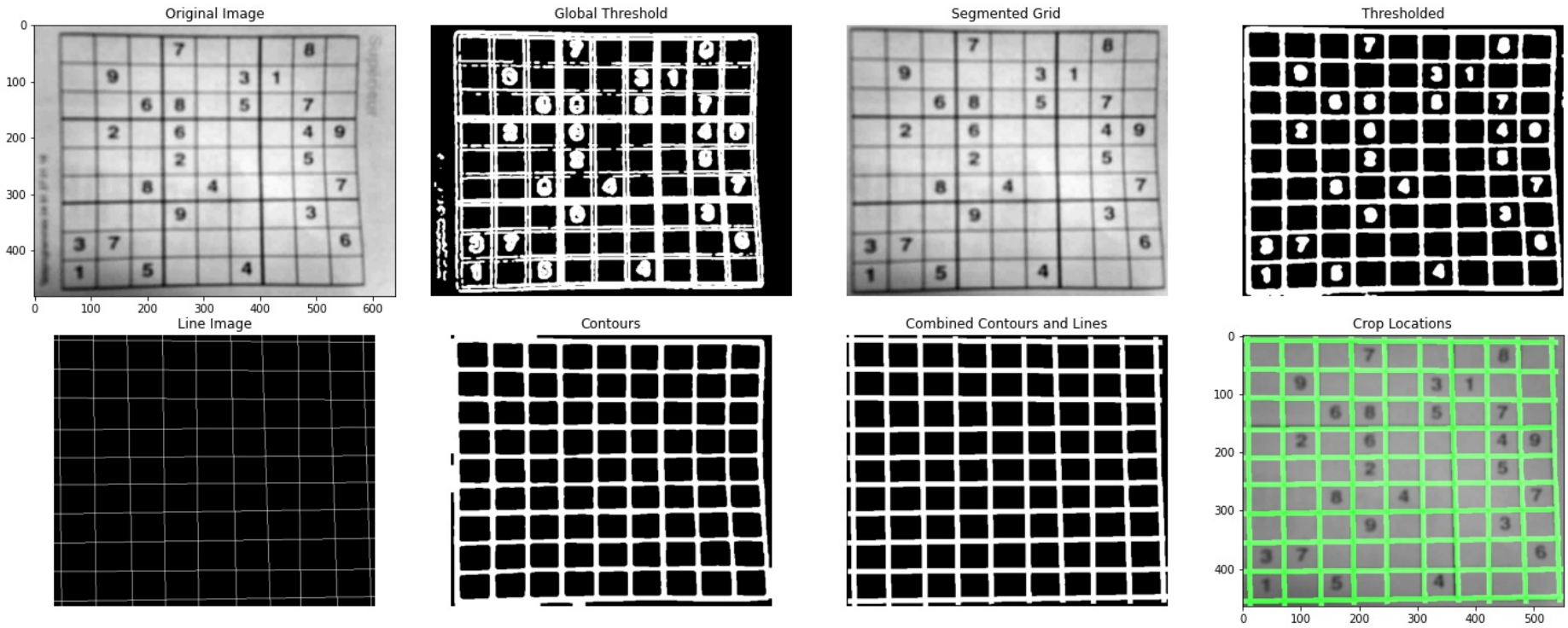


Figure: Intermediate Results.

Adaptive

We use a slightly adaptive method, by having multiple versions with slight differences. We use the simple version first, and gradually add complexity by, for example, dilating with rectangular structuring elements to repair edges, or altering how we combine the contour and Hough-line image.

Recognition Method I

Once we have the cropped cells, how do we classify the digits themselves?

1. We use the thresholded image to crop out the cell itself, as in 4b.
2. Use morphological labelling to extract the connected components, like 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 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 4e.
5. We then resize the image to a constant 16×16 using bi-linear interpolation (4f) and flatten it into a 256 dimensional vector.

Recognition Method II

There are many different methods that we can use to recognise the digits like K-Nearest Neighbours (KNN), KMeans and Principle Component Analysis (PCA), but Linear Discriminant Analysis (LDA) proved to be the best.

This works by trying to maximise the ratio of the between-class variance to the within-class variance [Balakrishnama and Ganapathiraju, 1998]. We use the previously generated 256-D vectors to classify the digits.

Training data was obtained by manually labelling some (235) cropped digits. We used about 175 digits as our training set and the other 60 as our testing set.

Digit Recognition I

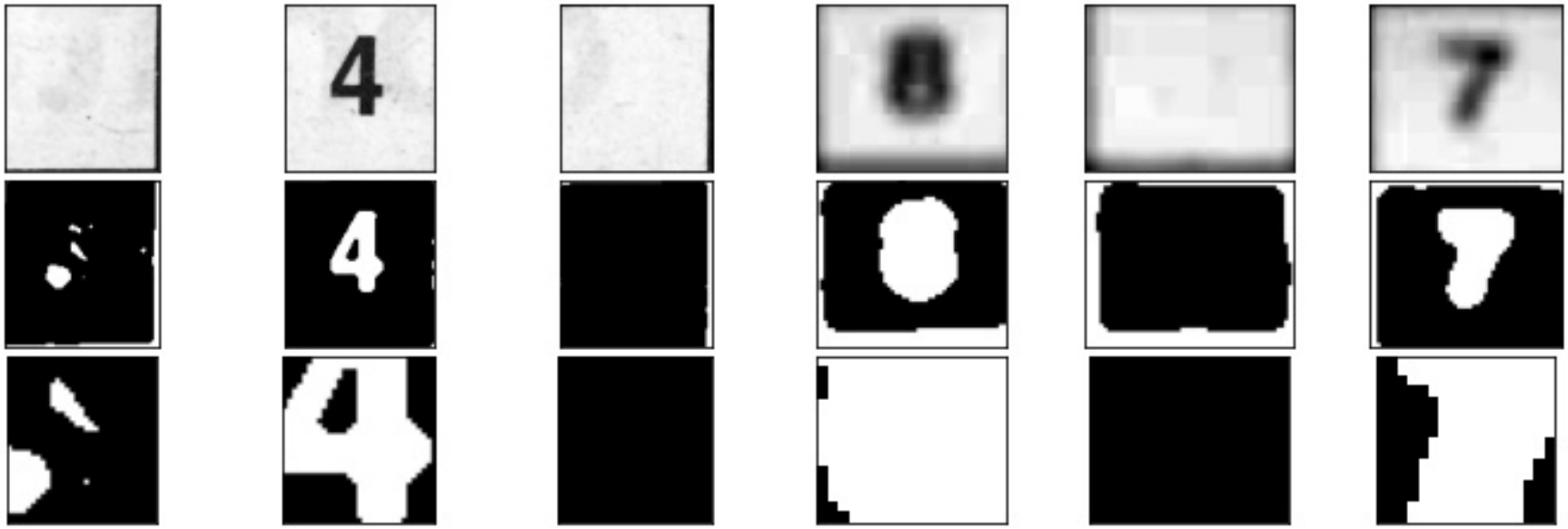
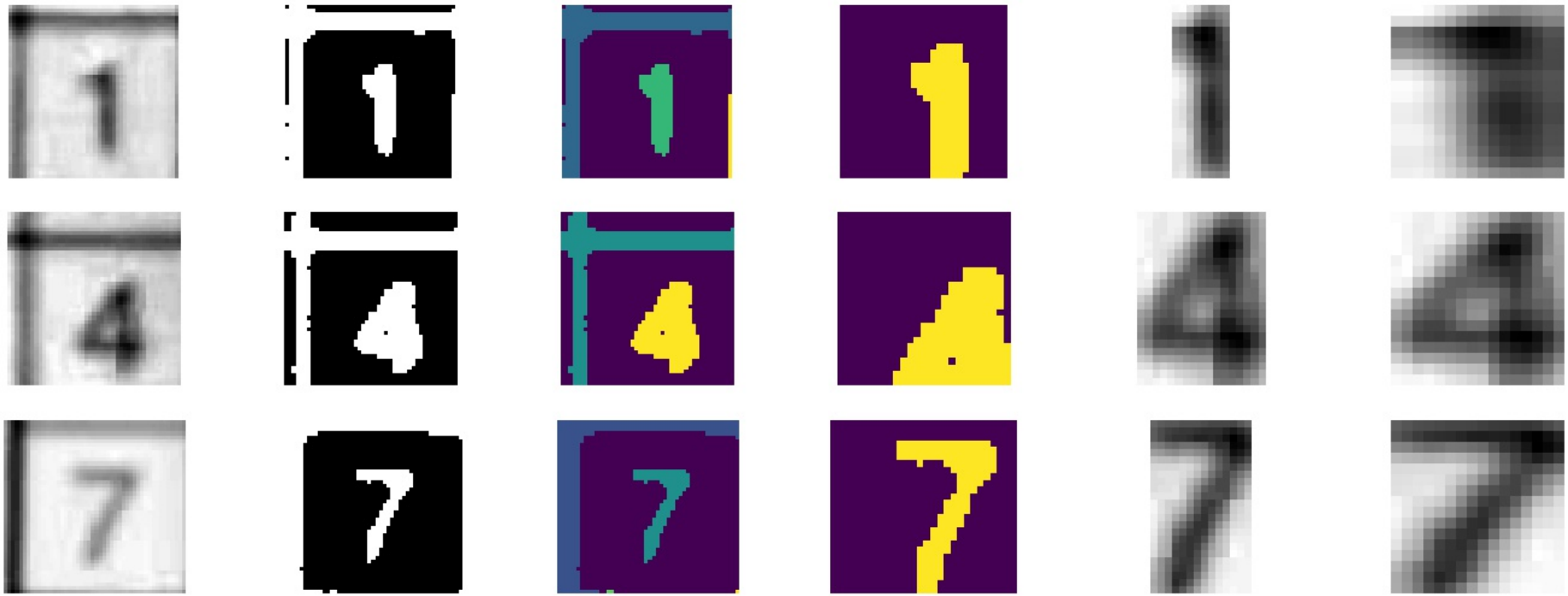


Figure: Empty and non-empty cells from two different grids. The first row is normal, the second row is thresholded and the bottom row is only the middle ninth of the image.

Digit Recognition II



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

Figure: Digit Recognition Steps

Digit Recognition III

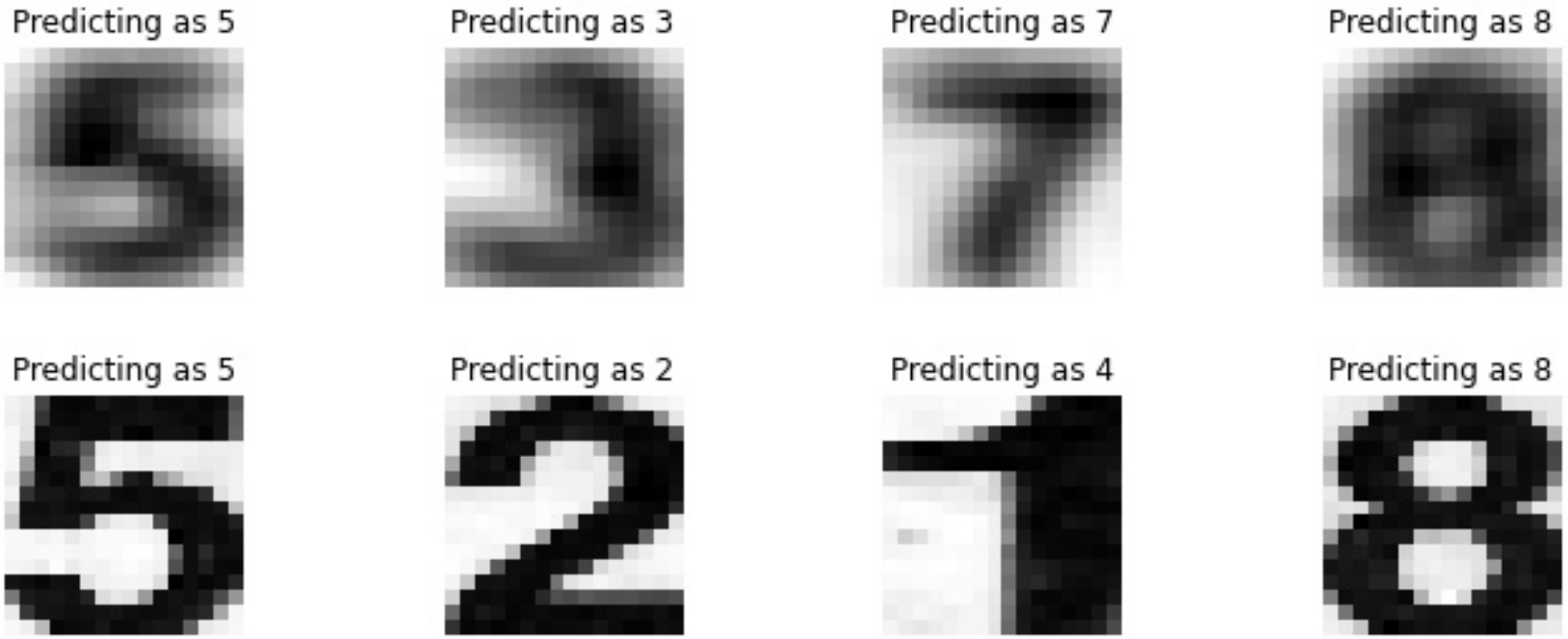


Figure: Some predictions

Overall Results

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

Results:

- Failed on 17% of images and produced invalid results.
- Of the remaining 83%, 93% predicted the locations of the digits correctly.
- 68% of non-failed grids were predicted perfectly.
- Of all individual digits, the method predicted 94% correctly.

Overall then, 78% of locations were predicted correctly and 56% of the time, we predicted the entire grid perfectly.

Overall Results II

Figure: Actual

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			

Table: Predicted

Failure Cases I

The method was not full proof, and here are some reasons why it failed:

1. Vastly different image styles/lighting conditions.
2. Images captured different components, multiple grids.
3. Digit recogniser hasn't been trained on those specific styles of digits.

Failure Cases II

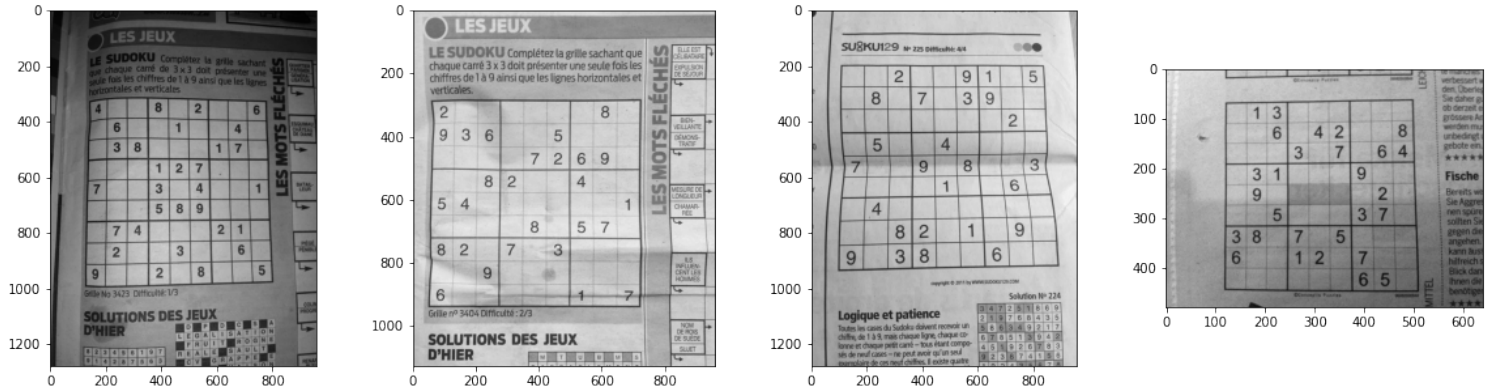


Figure: Original Images

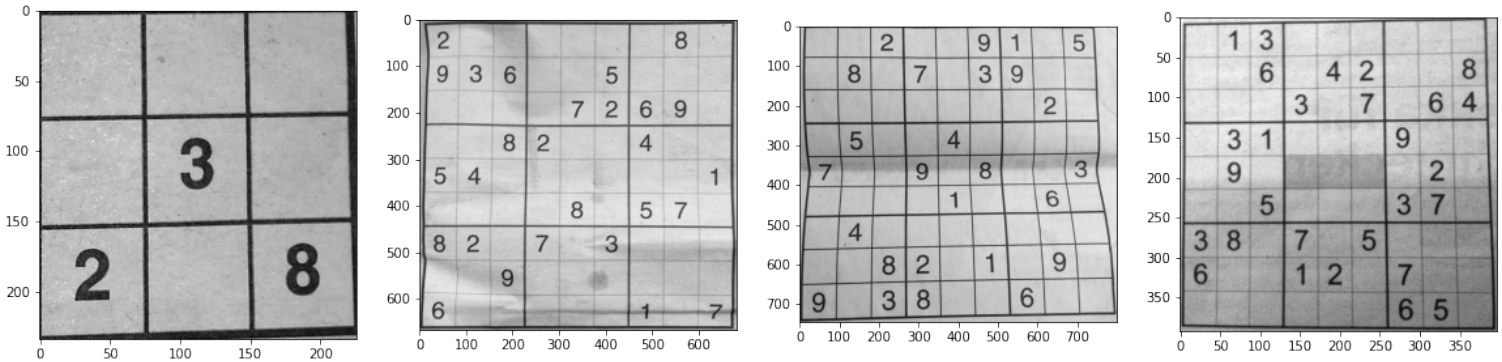


Figure: Segmented Grids

Failure Cases III

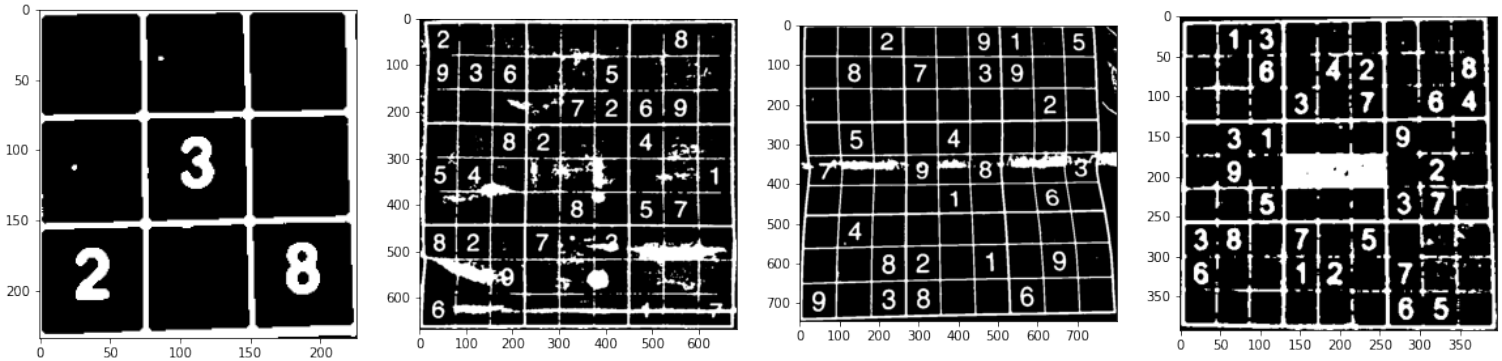


Figure: Thresholded Grids

Conclusion

In conclusion, we developed a relatively good method, that maps from images of sudoku grids to text files representing those grids that can easily be solved using off-the-shelf libraries.

The method was accurate, but it lacks in robustness to image conditions and other image styles, which could be improved in future work. Other future directions could be to improve the recognition method itself by either using a different method, or using more training data.

Thank You.


Balakrishnama, S. and Ganapathiraju, A. (1998).
Linear discriminant analysis-a brief tutorial.
Institute for Signal and information Processing, 18(1998):1–8.


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