Sudoku Extractor

William Hill (2115261)
Ireton Liu (2089889)

June 20, 2022

Abstract

Computing the solution to a Sudoku puzzle algorithmically with a computer when given the numbers and their corresponding coordinates in the grid is a trivial task. However, in many practical cases, these Sudoku puzzles are often presented in printed media. In these cases, in order to solve them using a computer, it is vital to be able to extract necessary information from these media through the means of photographs. This project presents a method of extracting the grid data from certain types of noisy and blurry photographs of Sudoku through the use of different image processing techniques.

1 Introduction

Sudoku puzzles, in their modern form, have been commonplace for the last four and a half decades. Around the time of their rise in popularity, the first personal computers were being manufactured and, with them, the desire to use computers to solve problems and puzzles such as sudoku, for both fun and research purposes. Consequently, many programs have been developed which take a text-based sudoku grid as input and solves it optimally.

However, it becomes tedious to manually convert a sudoku problem to its text form and, with recent advancements in good-quality cameras on personal devices, such as smartphones and tablets, it is a logical progression to simply be able to take a photograph of the sudoku grid and have a program return the text-based grid, which can then be passed to one of the existing solvers.

Consequently, we propose and implement a non-learning approach for detecting the digits and their correct location in a picture of a sudoku grid, and cropping them out. We further use the current state-of-the-art learning solution for recognising handwritten digits [1] to convert these cropped-out digits to text, and then output a text-based grid.

Notably, our method is relatively robust to various noise in the images, considering the variability in people taking the photographs in the first place: we are able to overcome different grid colours, crumpled grids, and blurriness, although only to a certain degree for the latter.

2 Terminology

Sudoku Box refers to the small box which may contain a digit in the sudoku puzzle

Sudoku Grid refers to the main grid containing the 81 Sudoku Boxes

3 Methodology

We have implemented a non-learning approach to detect and extract the boxes containing digits in the sudoku grid, followed by passing those extracted digits through a Convolutional Neural Network to convert them to text. The non-learning approach uses mathematical morphology quite heavily. The main stages of our approach are outlined below:

Note: any techniques marked with a * are algorithms taken from the course.

3.1 Detect and Extract the Sudoku Boxes containing Digits

Convert to single-channel image Initially, we convert any input image to a single-channel greyscale image, followed by a data-type conversion to integers. Converting the input image to greyscale is trivial, and allows us to process images taken by modern cameras, which store images as RGB images. Additionally, the data-type conversion to integers is an ease-of-use choice, allowing us to perceive and operate with the image in a discrete manner.

Convert to a binary image Mathematical Morphological operators are more-easily defined, used and interpreted for binary images. Considering we will make extensive use of these operators, we are required to convert the greyscale image to a binary image. We follow these steps:

1. Apply Gaussian filtering* to the greyscale image, using a standard deviation of 1, with the extent of the filter limited to four standard deviations. This performs a relatively small smoothing operation to remove any jagged or impulse noise introduced by the

camera, but leaves significant information, such as the digits and grid lines, largely unaffected. This smoothing operation also improves the effectiveness of later thresholding by reducing the variability in the image's histogram.

2. Apply Unmask Filtering* to enhance the grid lines and digits of the image, using a Gaussian Filter with a standard deviation of 1.5. When combined with the smoothing from the previous step, this has the joint effect of removing noise but increasing the clarity of the main objects in the image. **Note: the result of this operation is duplicated for later use (call it *image_1*).**

3. Perform local thresholding* with a neighbourhood of 151×151, to put the digits and grid lines in the foreground. We do not use a global method, such as Otsu's Method, due to background regions skewing the image histogram, thus resulting in a binary image with missing foreground information (either the digits and grid lines are blended with the background, or the grid boxes are set as foreground pixels).

4. Some further noise removal is performed on this binary image by performing an Opening* (with a 3×3 flat structuring element (SE)), followed by an Erosion* (with a disk SE with a radius of 2) to minimise any large foreground noise (considering the digits and grid lines are large, this extra erosion does not impact their distinctness).

Extract the main grid We perform an initial pass on the image to extract the vertical and horizontal grid lines, which together form the main sudoku grid and the sudoku boxes (with additional noise). From this, we detect the sudoku grid and crop the image to remove any background, so any further processing is not susceptible to the noise present in the background.

1. Perform two Openings* on the current image separately (i.e. the second opening is not performed on the output of the first opening). The first is done with a rectangular SE of size 41×1: the result is an image containing only the vertical lines of the Sudoku Grid and Boxes. The second is done with a rectangular SE of size 1×41: the result is an image containing only the horizontal lines of the Sudoku Grid and Boxes. The SEs were chosen this size in order to prioritise the lines of the sudoku grid and boxes and reduce the likelihood that background noise will be included, although some large lines from the background will inevitably be included. We then take the union of these two output images as the image of grid lines. This step is encapsulated in a function, `findGridLines`, and will be used again later.

2. Find the contours in the resultant image[1]. We approximate these contours by fitting a minimal Bounding Box (i.e. a rectangle) such that the contour is contained entirely within this box. This will identify all the boxes that occur in the image, including the main Sudoku Grid, the Sudoku Boxes, and any significant noise in the background.

3. Clearly, the largest bounding box (i.e. the one with the largest area) will correspond to the Sudoku Grid. We identify this box and use it to crop out the Sudoku Grid from the image saved after the unmask filtering (i.e. *image_1*) (this will yield a greyscale image of the Sudoku Grid).

Standardise the extracted Sudoku Grid Now that we have the grid isolated, we need to resize it so that later SEs used will be able to have a single size which works uniformly on all Sudoku Grids. This counteracts the variability in how people may take the initial image, but is based on the implicit assumption that the resized grid will have roughly the same scale in all the images (e.g. if the initial image had the grid far away, then the resized grid may be too small to yield effective results overall). We chose a resized image of 450×450 by experimentation. We perform a number of additional steps to this resized image:

1. Determine an indication of the sharpness of the image. The gradient magnitude* is computed for each pixel, and the average gradient magnitude is used as the sharpness score. A larger sharpness score is indicative of a sharper image, meaning its edges are more clearly defined. This score is used to aid the construction of SEs later, where different sizes are required to perform optimally depending on the sharpness of the image.

2. Perform a Histogram Equalisation* on this resized image to apply correction to the intensity distribution, which has the benefit of making later thresholding more accurate. **Note: the result of this operation is duplicated for later use (call it *image_2*).**

Convert the Sudoku Grid to a binary image The extracted and equalised greyscale image needs to be converted to a binary image again. This thresholding process will be more accurate considering the background of the image has been removed, thus the associated noise has been removed, leaving the digits and grid lines to be processed far more accurately.

1. Perform a Laplacian of Gaussian (LoG)* filter, with a standard deviation of 2, to identify the edges in the

[1]finding contours is an extension of finding Connected Components, where a component may contain many contours and where a contour is the group of connected pixels surrounding a hole, or background region.

image on a particular scale (defined by the standard deviation). This scale was determined by experimentation. The advantage of this step is that we enhance the digits and grid lines, whilst leaving any residual noise as is, thus making it easier to threshold the image and not have the noise interfere with the process of determining the correct threshold value. We want as accurate a binary image as possible so that we can accurately determine where the digits lie.

2. Perform global thresholding using Li's Method[2]

Extract the grid lines The grid lines of the Sudoku Grid are extracted again:

1. Perform a dilation* (with a disk SE of radius 2) to increase the prominence of the grid lines
2. Use the `findGridLines` method (described previously) to extract the grid lines of the Sudoku Grid.
3. Perform two Closings* on the extracted grid lines separately (i.e. the second closing is not performed on the result of the first closing). They are done with rectangular SEs of size 4×31 and 31×4 to fill in any gaps in the horizontal and vertical lines, respectively. **Call this *image_3*.**
4. Find the contours in the resultant image and approximate them by fitting a minimal Bounding Box (i.e. a rectangle) such that the contour is contained entirely within this box. This will identify all the boxes that occur in the image, including the Sudoku Grid and Boxes, and residual noise.

Determine the bounding boxes corresponding to the Sudoku Boxes Given the bounding boxes from the previous stage, we need to determine which of these actually corresponds to the 81 boxes that occur in the Sudoku Grid so that we can determine which have digits, and extract them.

1. Filter the bounding boxes so that only squares (or approximate squares) remain. We pass a bounding box if the ratio between the vertical and horizontal lengths (the smaller of the values to the larger of the values) is larger than a threshold, which we determined to be 0.6. This is based on the idea that a square has the same length sides, so their ratio would be 1 and as this ratio decreases, we get rectangles instead.
2. Sort the passed bounding boxes by their area, in ascending order, so that small squares appear first, then the Sudoku Boxes, and then large squares, where the small and large squares are from noise (and the largest is the main Sudoku Grid).
3. Take the median box from this list, and use its area as an indication of the "correct" area of the Sudoku Boxes.
4. Sort the boxes again by the difference of their area from the "correct" area, in ascending order. This puts all the boxes closest to being the Sudoku Boxes at the front of the list.
5. Take the first 81 boxes from the list as the Sudoku Boxes.
6. Sort these 81 boxes so that their position in the list corresponds to their linearised index (in row-major order) of their position in the Sudoku Grid. This is done by first sorting the boxes by their top y-coordinate (i.e. their vertical position) to put the boxes into groups by row. Then sort each consecutive group of 9 boxes (i.e. each row) by their left x-coordinate (i.e. their horizontal position) to put each box into the correct column (or position in the row).

Determine which Sudoku Boxes contain digits Now that we have the 81 Sudoku Boxes and in their correct order, we determine which of the boxes contain a digit, since only these boxes need to be extracted and saved for digit-recognition.

1. Perform an erosion* on **image_3** (of the extracted grid lines). The SE used will be based on the sharpness score calculated earlier. If the sharpness is below 0.03, use a square SE of size 6, otherwise use a square SE of size 5. This reduces the size of the digits by an amount based on the image sharpness such that they remain significant, but also ensures that any noise in the Sudoku Boxes is also minimised.
2. Crop out each Sudoku Box (after first reducing the size of the box with padding of 5 pixels all around) and detect whether a digit is present in this cropped-out box. This is done by taking the ratio of foreground pixels to total pixels in the box, and square rooting the answer (so that the number with which we have to work are larger and require less granular fine-tuning). We say that a digit is present if this value is larger than a threshold, which we determined to be 0.22.
3. If a box has a digit, crop the box from the equalised greyscale image (i.e. **image_2**), and save the image (with the filename as the linear index of this box in the grid).

3.2 Convert the Extracted Digits to Text

We will use a pretrained Convolutional Neural Network [1], which has high accuracy on the MNIST Dataset, for this stage, where we convert the digits cropped out in

[2]Li's Method minimises the cross-entropy between the foreground and foreground mean, and the background and background mean.

the previous stage to text, which can be used as input to a Sudoku Solver program. The CNN is only an approximate solution to our digit recognition since it was initially built for a different purpose. However, since the different fonts and orientations can be approximated to handwriting styles, we determined that it was sufficient for our purpose.

Convert the digits to the correct format The pretrained model requires a 28×28 image as input. We resize each box image to this size. Additionally, we apply Li's Thresholding to the resized image since the binary image yielded far greater accuracy (overall) from the model, for our intended use.

Convert the digits to text Each formatted box is then passed to the pretrained model to obtain a prediction of the digit represented in the image. The model actually returns a list of the possible digits in order of decreasing accuracy. For some predictions, it thinks 0 is the most likely digit, which is illegal in a Sudoku puzzle, so we use the second most-likely digit in these cases.

4 Experimental Setup

The dataset used for images of Sudoku is provided by Baptiste Wicht on GitHub [2]. This dataset contains photographs of Sudokus with many varying properties, such as dimension, clarity, lighting conditions, noise level, etc... We used a subset of the dataset to test our method against these photographs with drastically different properties. Here, a few of these images are selected and discussed.

Figures 1, 2 and 3 are examples of images from the subset that display drastically different characteristics. Figure 1 is an image with moderately high definition, where although there is still blurriness, numbers and lines are clearly visible and distinct, however, the lighting across the image is inconsistent. Figure 2 is an image that is, on top of having uneven lighting, very unclear in different areas. Figure 3 shows a high definition photograph with sharp and distinct edges across the image, however, it contains a lot of unnecessary information surrounding the Sudoku grid itself. Using these images, the robustness of the method is tested by answering the following questions:

1. How successfully will our method perform for these images with varying characteristics?
2. If our method fails, at what stage of the process did it fail?

The CNN by Sanghyeon An [1] was used to perform digit extraction. The model is trained on the MNIST handwritten digit dataset [3] until it had a 99.7% test accuracy, this was chosen due to the lack of availability of datasets that contain printed digits. For this reason, we expect discrepancies between the expected output and the actual output of our method.

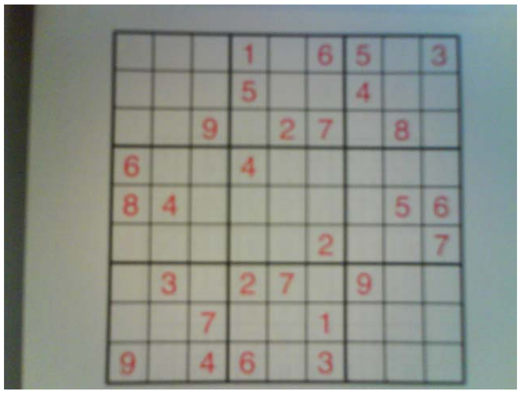


Figure 1: Example of a standard image of Sudoku with uneven lighting.

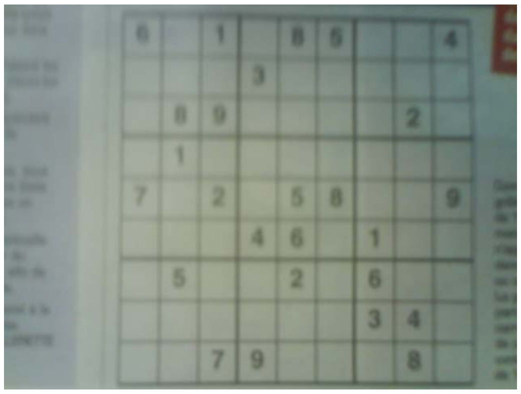


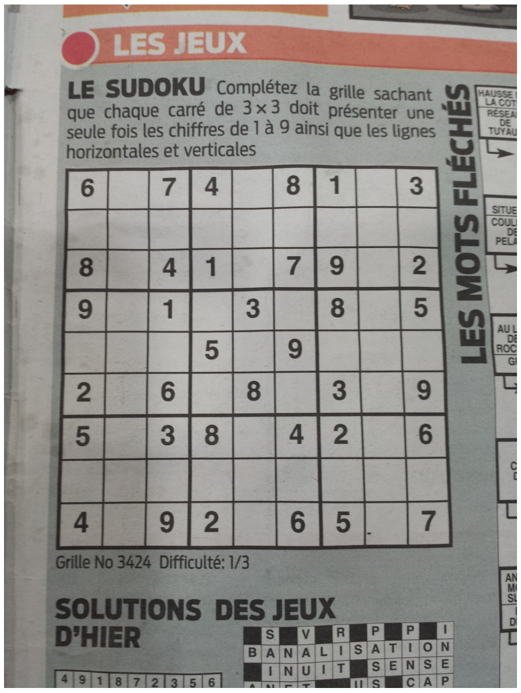
Figure 2: Example of a partially illegible image due to blurriness.

5 Evaluation and Discussion

In this section, the results produced by our method for different inputs as well as the intermediate processing on the image are evaluated. Here, we looked at two successful examples with different artefacts and discussed the effectiveness of our method as a whole, as well as how each of the individual steps was successful in removing the undesired information from the image. We also looked at two different cases of failures and discuss the flaws in our method that caused the failure, as well as potential fixes that can be implemented to account for these failures.

5.1 Results

Figure 4 shows a sequence of images that corresponds to different stages of our method described in section 3. The



LE SUDOKU Complétez la grille sachant que chaque carré de 3×3 doit présenter une seule fois les chiffres de 1 à 9 ainsi que les lignes horizontales et verticales

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

Grille No 3424 Difficulté: 1/3

SOLUTIONS DES JEUX D'HIER

Figure 3: Example of a high quality image with a lot of extra information.

input image, as described in section 4 has uneven lighting. It can be seen from the thresholded image (b) that using adaptive thresholding can clearly separate the foreground elements (the Sudoku puzzle) and background elements, given that the difference was already distinct enough. The method then accurately extracted the grid (c) from the rest of the image, thus removing all the unneeded information from the process. Through the use of specific morphological operators described in section 3, the method is able to clearly extract and emphasise the grid lines of the puzzle (d), this helped in identifying the precise bounding boxes for each cell of the Sudoku, which allowed for detection and extraction of individual digits (e). Figure 5 shows the first four numbers successfully extracted from the original image.

Image (f) shows a screenshot of the output our method printed to the terminal. It is clear that meaningful digits were successfully extracted from all the regions where a number can be seen in the original image. Our method identified and extracted the exact number of digits present in the input image (28), out of which 26 of them were correctly predicted (93% accuracy). As mentioned in section 4, the inaccurate predictions are likely due to the fact that the neural network used was trained on handwritten digits. More specifically, it can be seen that for this specific input image, the only digit that was incorrectly extracted was the digit "6", which was incorrectly, but consistently predicted as "8".

We speculate the reason for the error as follows: in handwritten digits, the digit "6" is most often written with a much straighter upper half than its printed counterpart presented in the input image, at the same time, the bent curve in the printed digit "6" resembles the top half of the digit "8". This only serves as an explanation for the errors in this case, however, other errors in the result of different input images can be explained with reasons of a similar nature.

Therefore, we consider our method successful in extracting the Sudoku for this input image. On top of obtaining a very high accuracy, the recognition of the number does not form the main part of our method, as we can easily interchanged it with a more suitable method for predicting printed digits, given the availability of the data.

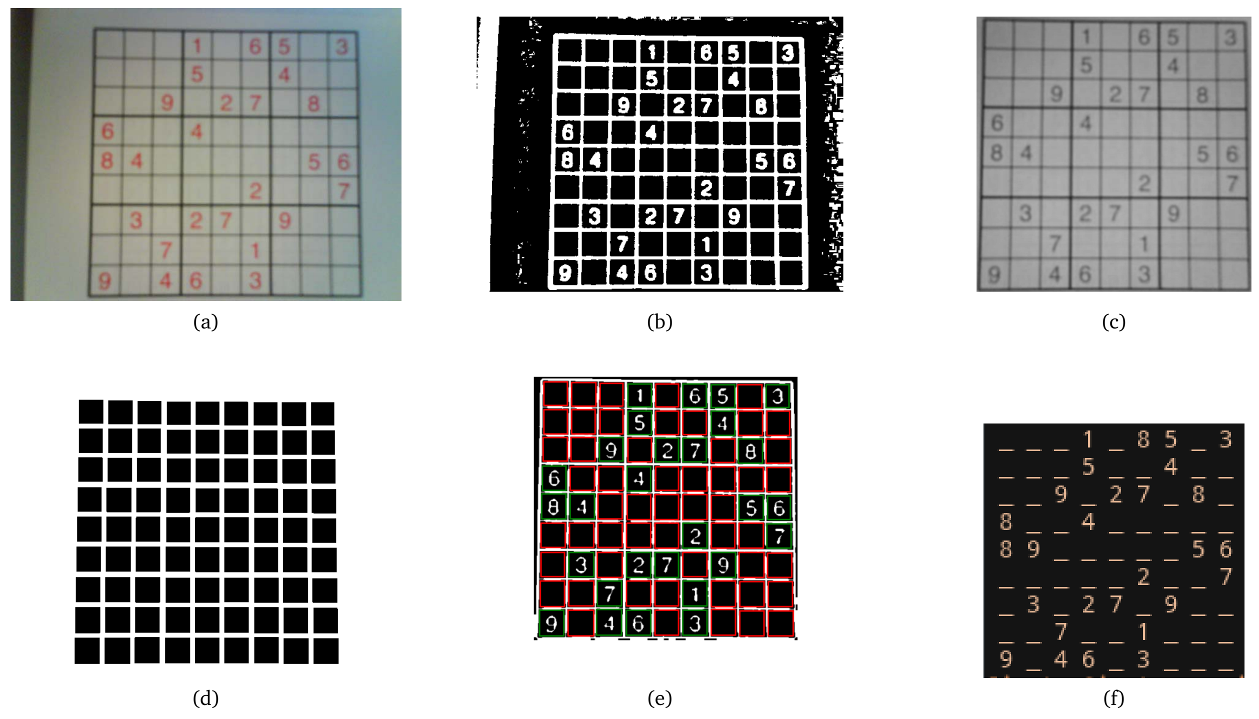
Figure 8 shows other examples of successful extractions on other input images with different characteristics. Image (a) is a low definition photograph of a Sudoku puzzle taken under bad lighting conditions. In this case, our method is still able to correctly identify all the grid lines and extract all the regions in the image that contains a digit. However, due to the poor quality of the image, coupled with the reasons described above, the neural network was unable to accurately predict the correct digits (25% accuracy). Image (b) shows a photograph of a Sudoku puzzle that is surrounded by text and figures that are unnecessary to the problem, and can be considered noise. Our method is able to successfully isolate the Sudoku grid from the rest of the image and produce an output that is 97% accurate. From this, we logically assume that the accuracy of the output text is also heavily dependent on the clarity of the input image.

5.2 Shortcomings

There are a few conditions under which our method will fail at extracting the digits. In general, poorly captured photographs such as the one shown in figure 2, where lines are not clearly defined due to the blurriness of the image. In these cases, our method will fail to extract the precise grid lines from the image, unlike figure 4(d). In the section, we will discuss some shortcomings and limitations specific to our method, as well as potential solutions to them if it exists.

As our method assumes minimal rotation on the Sudoku grid in the image, any images where the Sudoku is visibly rotated (see figure 7) will cause our method to fail. One approach to account for the rotation of the image is to apply Hough transform to the image. By using Hough transform, the gradient of the most frequently occurring grid lines can be used to predict the rotation in the image, and its orientation can be adjusted as a pre-process. However, this solution is limited to rotations $< 90°$. Currently, we have not found a potential solution for rotations $\geq 90°$ without manually adjusting the image.

Another limitation of our method can be seen in figure 6. In this input image (a), a few coloured cells can be seen at certain locations within the Sudoku puzzle. When the grey scale pixel intensity of these coloured cells ap-



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

Figure 4: a) Original image. b) Thresholded image with local thresholding. c) The Sudoku cropped out of the original image. d) Extracted and emphasized grid lines of the Sudoku. e) Bounding boxes of each cell, colour coded by whether a number exists. f) Screenshot of the output text produced by the CNN

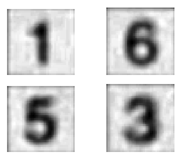
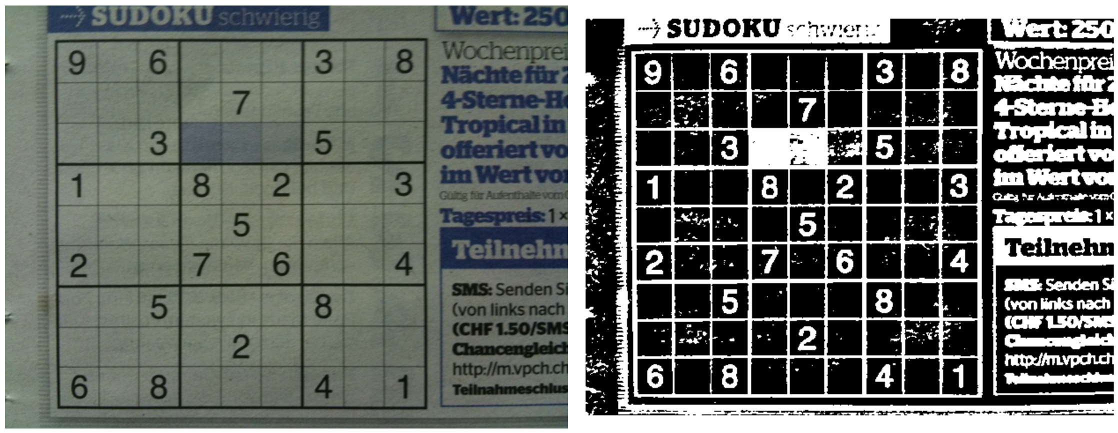


Figure 5: The first four digit cells extracted from the input image from figure 4

proaches the pixel intensity of the surrounding lines, the adaptive thresholding we perform will produce erroneous results, as can be seen in (b). We have not, however, found a good solution to solve this due to the specificity of this limitation to the image itself. This means finding a fix to this problem will potentially cause errors for other known working inputs.

6 Conclusion

In conclusion, our method is able to robustly and accurately detect and extract regions where the digits of a Sudoku puzzle lie, from a photograph of the Sudoku puzzle with varying characteristics and quality. The core of this method is to reduce the complexity of the problem by extracting just the Sudoku grid itself, from an image with surrounding noise. We then perform the necessary boundary detection and extraction operations to find the specific regions of interest. Our method is limited to images where the Sudoku is correctly aligned and oriented, with consistent colour filling within the cells of the grid.



(a) (b)

Figure 6: a) An Sudoku puzzle where a few of the cells are coloured in. b) Erroneous thresholding due to the cell colouring.

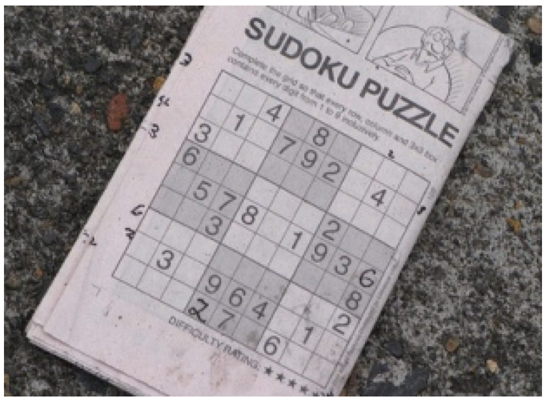
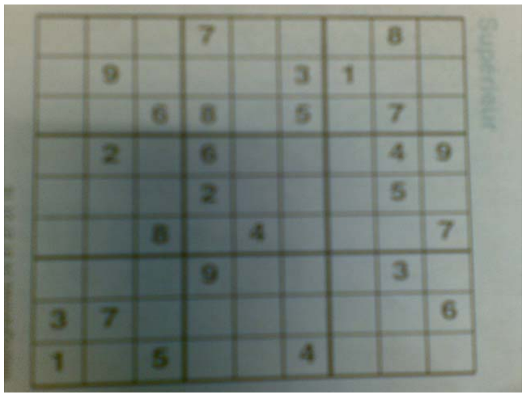
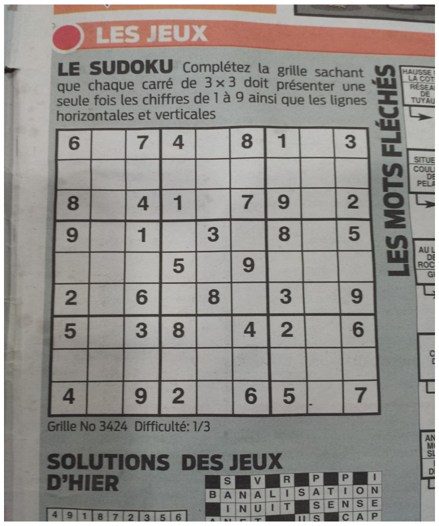


Figure 7: An image of a rotated Sudoku puzzle



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

(a)



LES JEUX

LE SUDOKU Complétez la grille sachant que chaque carré de 3×3 doit présenter une seule fois les chiffres de 1 à 9 ainsi que les lignes horizontales et verticales

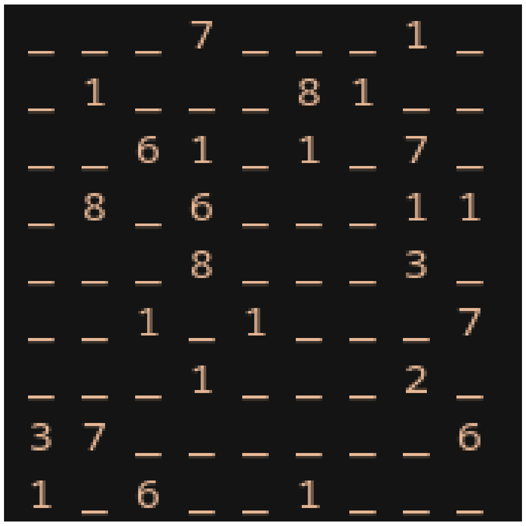
6		7	4		8	1		3
8		4	1		7	9		2
9		1		3		8		5
			5		9			
2		6		8		3		9
5		3	8		4	2		6
4		9	2		6	5		7

Grille No 3424 Difficulté: 1/3

LES MOTS FLÉCHÉS

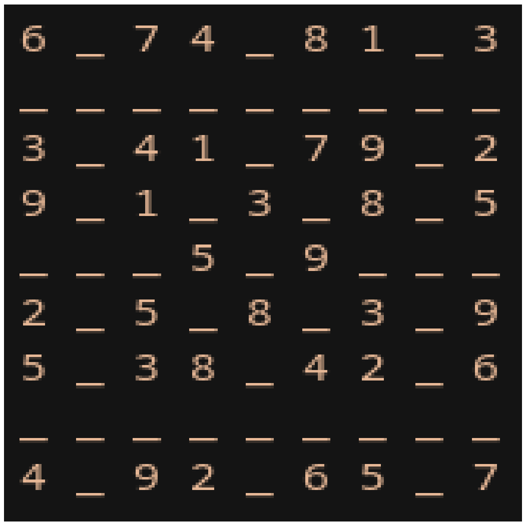
SOLUTIONS DES JEUX D'HIER

(b)



```
_ _ _ 7 _ _ _ 1 _
_ 1 _ _ _ 8 1 _ _
_ _ 6 1 _ 1 _ 7 _
_ 8 _ 6 _ _ _ 1 1
_ _ _ 8 _ _ _ 3 _
_ _ 1 _ 1 _ _ _ 7
_ _ _ 1 _ _ _ 2 _
3 7 _ _ _ _ _ _ 6
1 _ 6 _ _ 1 _ _ _
```

(c)



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

(d)

Figure 8: a) Blurry Image with bad lighting condition. b) Clear image with a lot of unnecessary information. c) Output of a). d) Output of b).

References

[1] S. An, M. J. Lee, S. Park, H. Yang, and J. So, "An ensemble of simple convolutional neural network models for MNIST digit recognition," *CoRR*, vol. abs/2008.10400, 2020. arXiv: `2008.10400`. [Online]. Available: `https://arxiv.org/abs/2008.10400`.

[2] B. Wicht, *Sudoku dataset*, https://github.com/wichtounet/sudoku_dataset, Accessed: 2022-05-30, 2019.

[3] C. J. B. Yann LeCun Corinna Cortes, *The mnist database of handwritten digits*, http://yann.lecun.com/exdb/mnist/, Accessed: 2022-05-30, 2010.