OCR by Team OT
First Report

Mathieu Dubrulle
Emile Preumont
Elias Tebbani
Assil Achour

October 2021

Contents

1 Introduction

This report gives an overview of the completed work on the **Optical Character Recognition** Project (OCR) by **Team OT**. In the scope of this project, an OCR for detecting and completing Sudoku grids is being made. To do so, the program starts by analyzing the image, then solves the Sudoku grid to finally display it being solved. This project is implemented in C and uses the SDL library. Files are manipulated in the .bmp format but our program can also analyse images of other formats.

2 Image loading and color removal

For this task, the IMG_Load function, that is pre-created by the SDL library, was used.

2.1 Grayscale

To apply grayscale, our image is parsed and is applied the following formula for the value r, g and b and for every pixel :
0.3*r + 0.59*g + 0.11*b;

2.2 Threshold

The main idea in our thresholding processing is that each pixel is compared to an average of the surrounding pixels.If the value of the current pixel is 35 percent lower than the average then it is set to black, otherwise it is set to white.

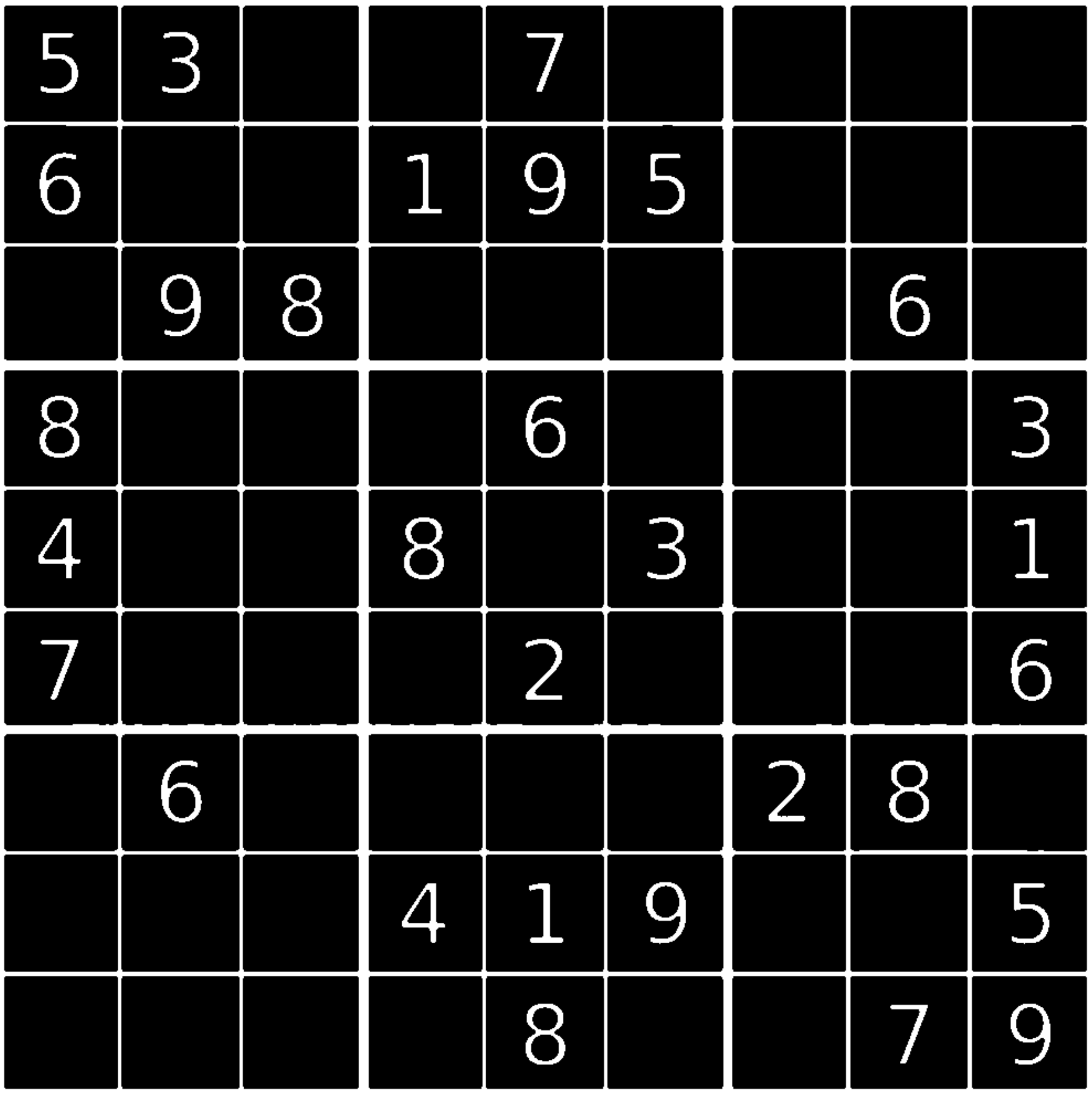
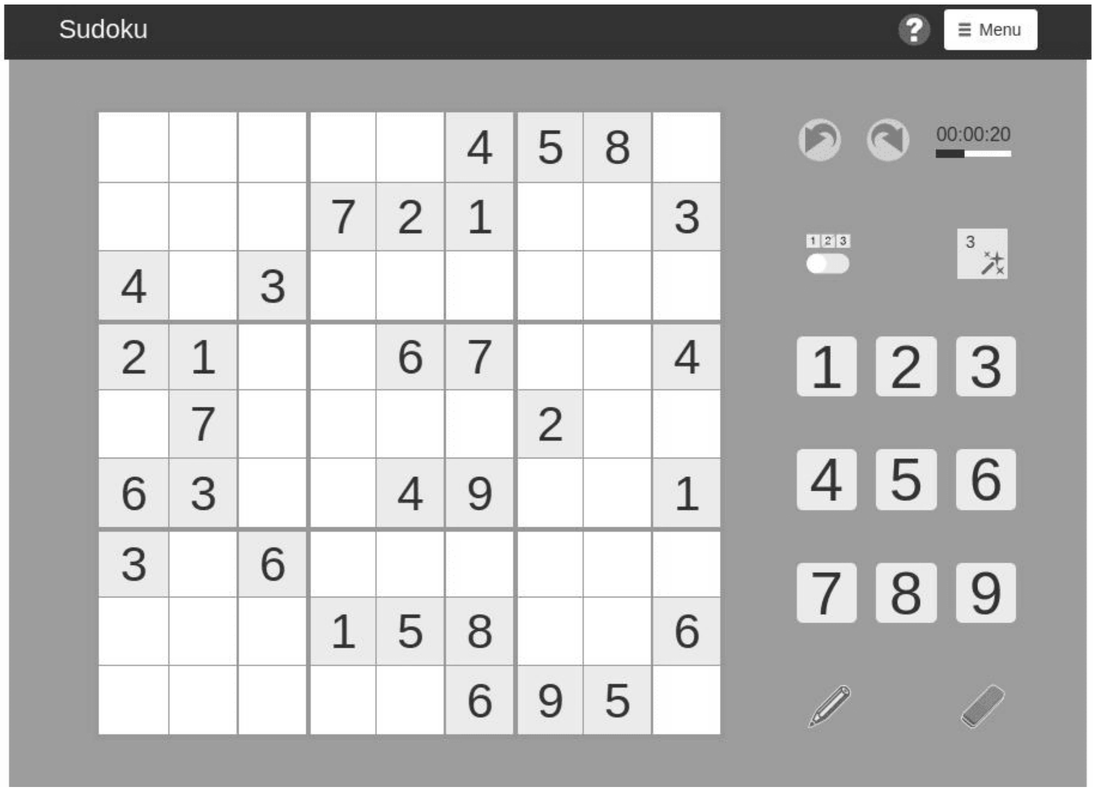


Figure 1: Current output of the adaptive threshold on gamma of image 1



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

Figure 2: Current output of grayscale filter on Image 3

3 Manual rotation of the image

For this task we are simply going to use the SDL_GFX library and use the the SDL_Rotozoom surface command. This command takes a degree of rotation as a parameter and rotate the image taking account of this degree.

4 Detection of the grid and the grid cell positions

4.1 Noise removing and blur

Gamma : In order to reduce the noise of the given pictures we will brighten the image a lot using gamma correction. The image will become so bright that most of the noise will be removed.

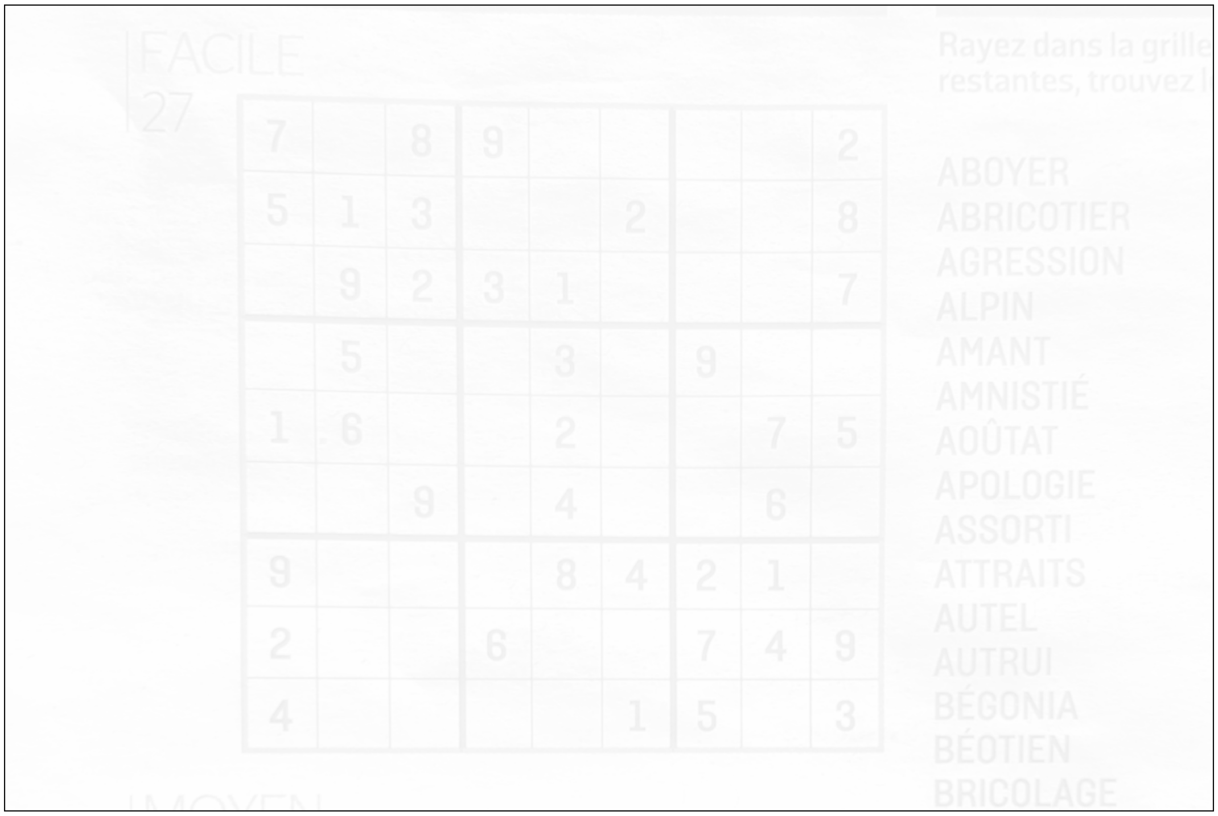
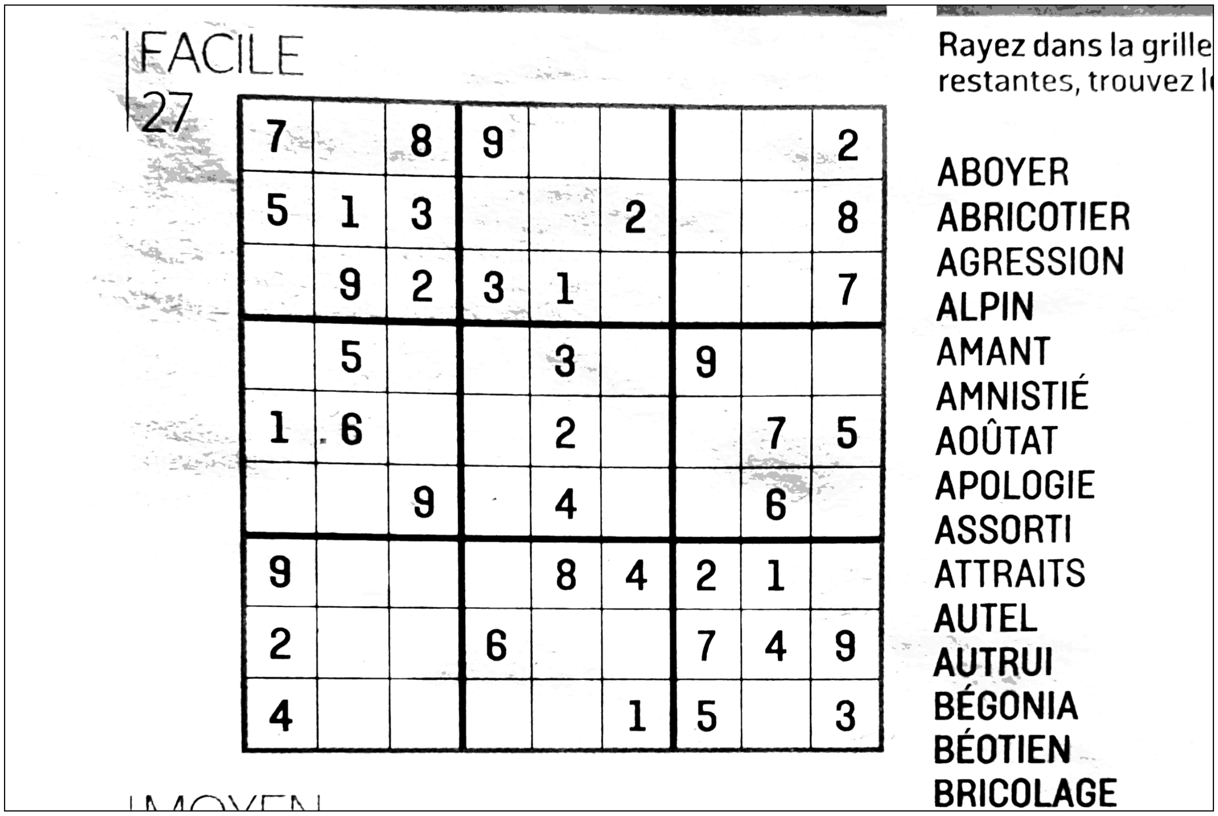


Figure 3: Current output of gamma on image 4

Contrast : Now that the image is really bright we need to reconvert into a "normal" image using a contrast correction factor:

$$\frac{259(C+255)}{255(C-259)}$$

C being the average value of the pixels of our image


FACILE
27

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

Rayez dans la grille
restantes, trouvez l

ABOYER
ABRICOTIER
AGRESSION
ALPIN
AMANT
AMNISTIÉ
AOÛTAT
APOLOGIE
ASSORTI
ATTRAITS
AUTEL
AUTRUI
BÉGONIA
BÉOTIEN
BRICOLAGE

MOYEN

Figure 4: Current output of contrast on gamma of image 4

Gaussian Blur : A Gaussian blur is applied to the image. It is applied using a mask (or convolution matrix) of 5 by 5 generated with a Gaussian function with the parameter gamma of 1,4. This filter reduces noise and removes small details of the image. This step prepares the next steps of edge and line detection. The matrix used :

$$\begin{bmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{bmatrix}$$

The result of the convolution is then multiplied by $\frac{1}{159}$ and obtain the following result :

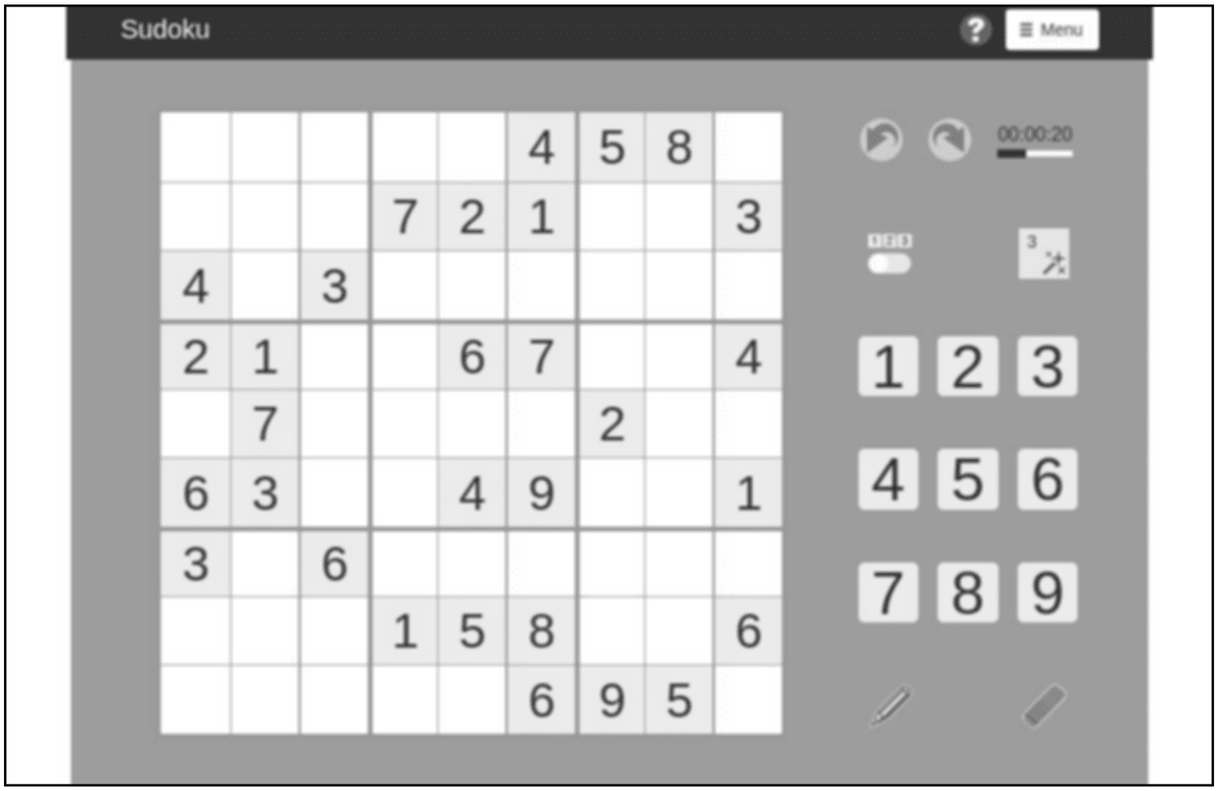


Figure 5: Current output of Gaussian filter on Image 3

4.2 Edge detection

Sobel Filter : The Sobel filter is used to detect the edges of the image. The Sobel filter uses convolution of two matrices to get a gradient of the pixel values calculated as follows :

$$\sqrt{(Gx^2) + (Gy^2)}$$

with :

$$\text{Gx} = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \text{ and Gy} = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

The direction of the pixel is calculated as so : Θ = atan2(Gy, Gx).
Sobel filter gives good results but the lines resulting are thick and so not perfect for line detecting. This is why we apply Canny after it.

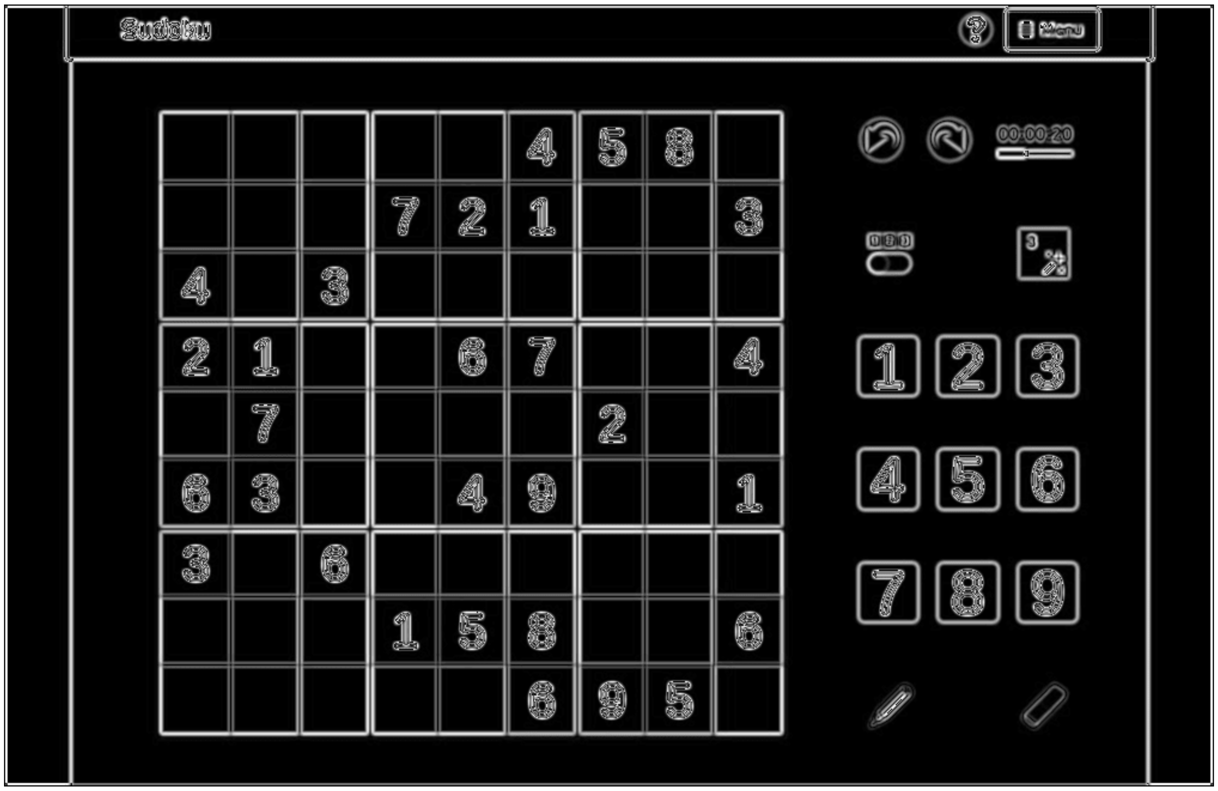


Figure 6: Current output of Sobel filter on Image 3

Canny Filter : Canny filter is used to detect edges and the resulting image has only 1 pixel wide lines. The first step is the lower bound cut-off suppression which removes the non-maximum on each lines. The second step is the double threshold and the Edge tracking by hysteresis which can be resumed as the analysis of which pixel are kept based on their connections and their gradient.

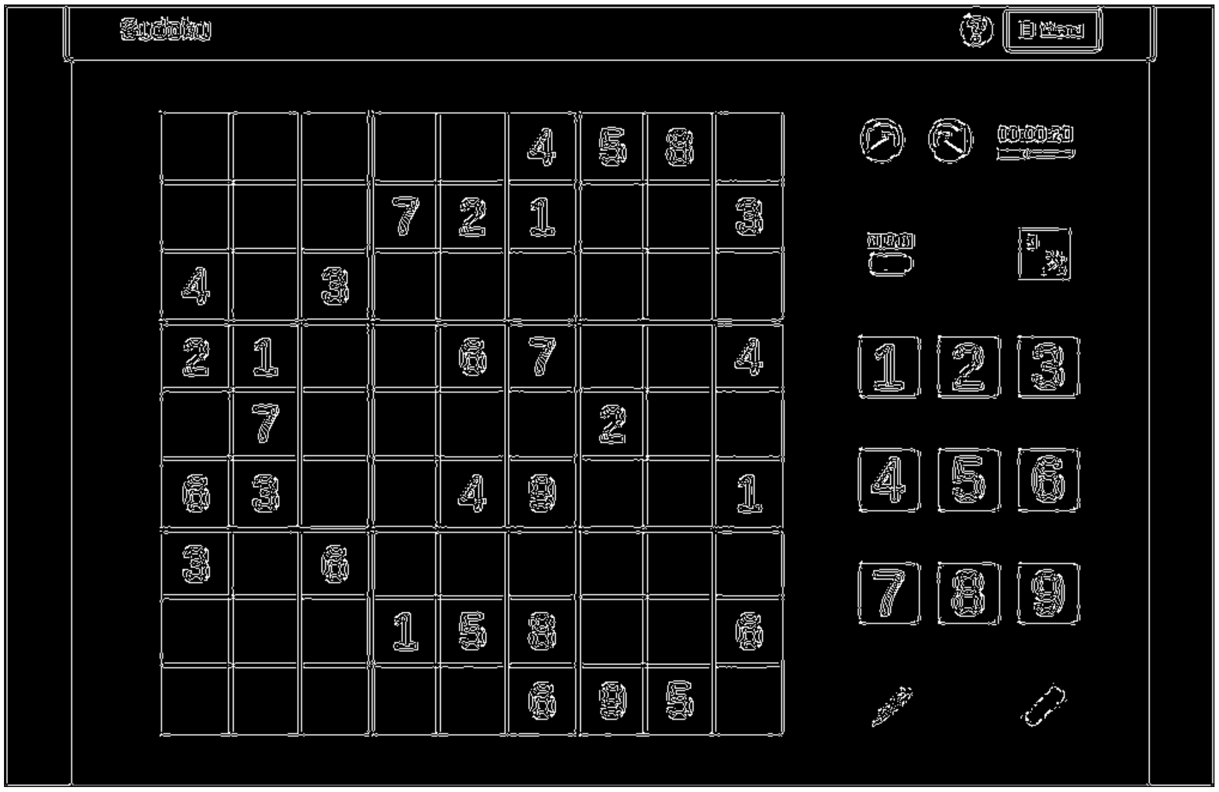


Figure 7: Current output of Canny filter on Image 3

4.3 Line Detection

Hough transformation : In order to simplify the detection of the Sudoku after the image processing, the Hough transformation algorithm has been implemented. This algorithm makes the edges of the image thinner and it works as follows:

1. Getting the edges pixel lines polar coordinates. Those pixel are recognisable thanks to previous image processing algorithms making them white.
2. Saving the number of occurrences of the lines into a Hough space which is a matrix with two coordinates, rho and theta from the line polar coordinates.
3. Selecting only the lines having high occurrences in the Hough space and drawing them. Those lines are the new edges which will be used in the Sudoku grid detection later on.

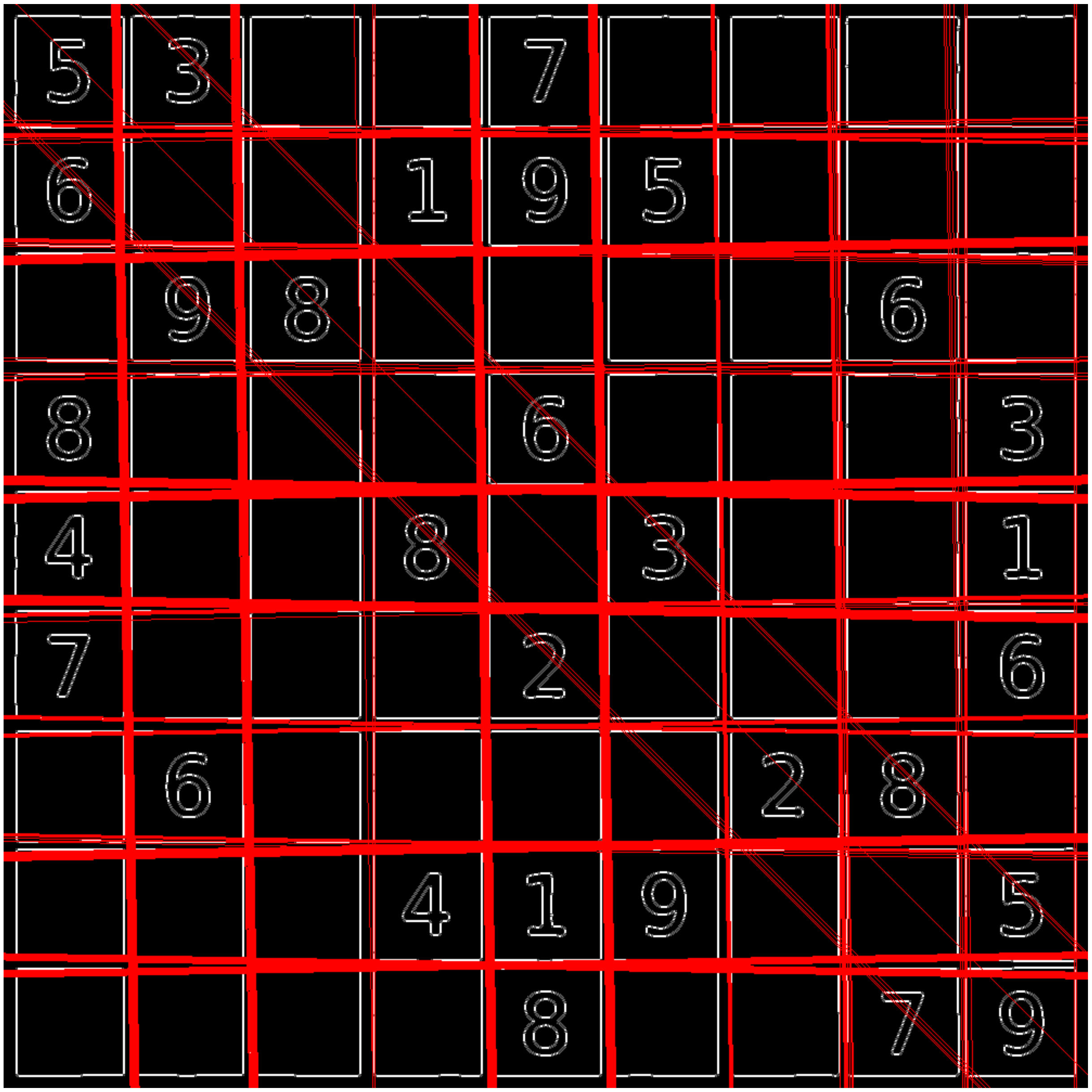


Figure 8: Current output of Hough transform on Image 1

5 Splitting of the picture

A function invoked with the coordinates of the top left pixel of the Sudoku grid and the width and the height of the grid allows one to split the image in 81 sub-images corresponding to the cells of the grid respectively. It uses **this function**.

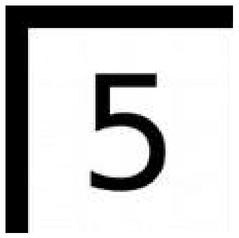


Figure 9: first cell of first image given by splitting

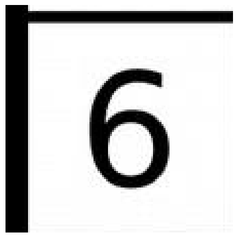


Figure 10: another cell

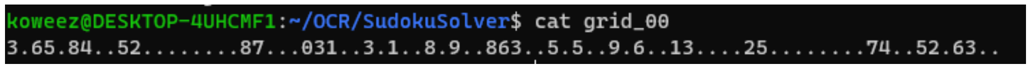
6 Implementation of the sudoku solving algorithm and save and load

6.1 Sudoku Solving algorithm

The sudoku solver algorithm uses backtracking. Basically it works making assumptions on the possible solution of the sudoku. It iterates through the grid and make an assumption of a legal value at the first non-empty cell. Then it follows this assumption until realising it does not work and tries another value. This mechanism goes on until the soduku is solved.

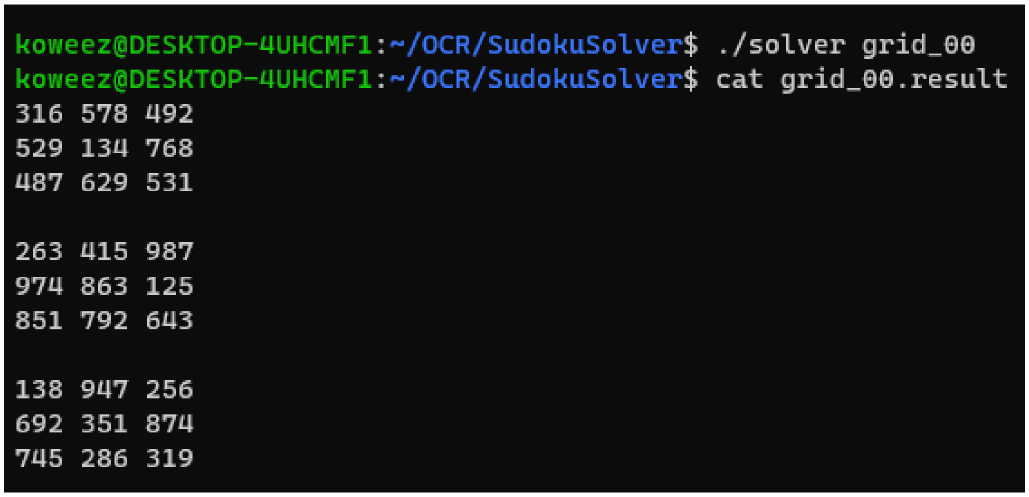
6.2 File manipulation and Save and Load of the results

As a result of the whole image processing part, the Neural Network returns a series of numbers corresponding to the sudoku grid. Those numbers are saved in a file which will be retrieved for the sudoku solver algorithm which will itself return another file corresponding the solved sudoku grid. As for now the neural network is not ready yet to work on numbers, a file with an unsolved sudoku grid has been created to work as the input of the sudoku solver. The input file is specified as an argument of the main and the output file is created in the program.



```
koweez@DESKTOP-4UHCMF1:~/OCR/SudokuSolver$ cat grid_00
3.65.84..52........87...031..3.1..8.9..863..5.5..9.6..13....25........74..52.63..
```

Figure 11: input file



```
koweez@DESKTOP-4UHCMF1:~/OCR/SudokuSolver$ ./solver grid_00
koweez@DESKTOP-4UHCMF1:~/OCR/SudokuSolver$ cat grid_00.result
316 578 492
529 134 768
487 629 531

263 415 987
974 863 125
851 792 643

138 947 256
692 351 874
745 286 319
```

Figure 12: output file

7 Neural Network

7.1 The XOR Network

A first neural network has been made : a **XOR** network. The back propagation aspect of this network uses the **Stochastic Gradient Descent**. It has two input layers, two hidden layers and one output layer.

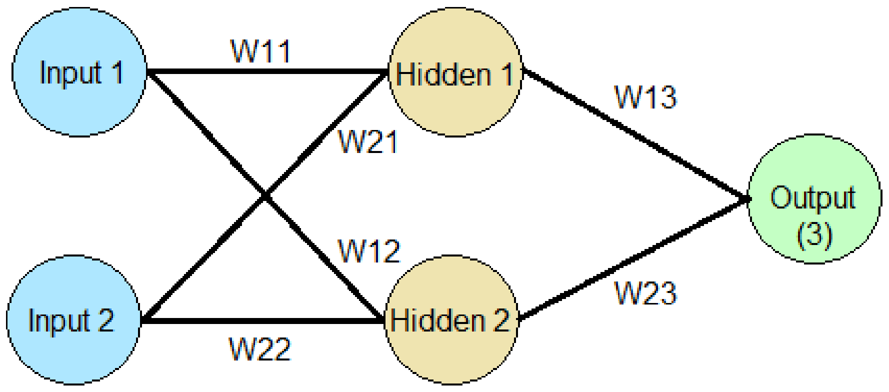
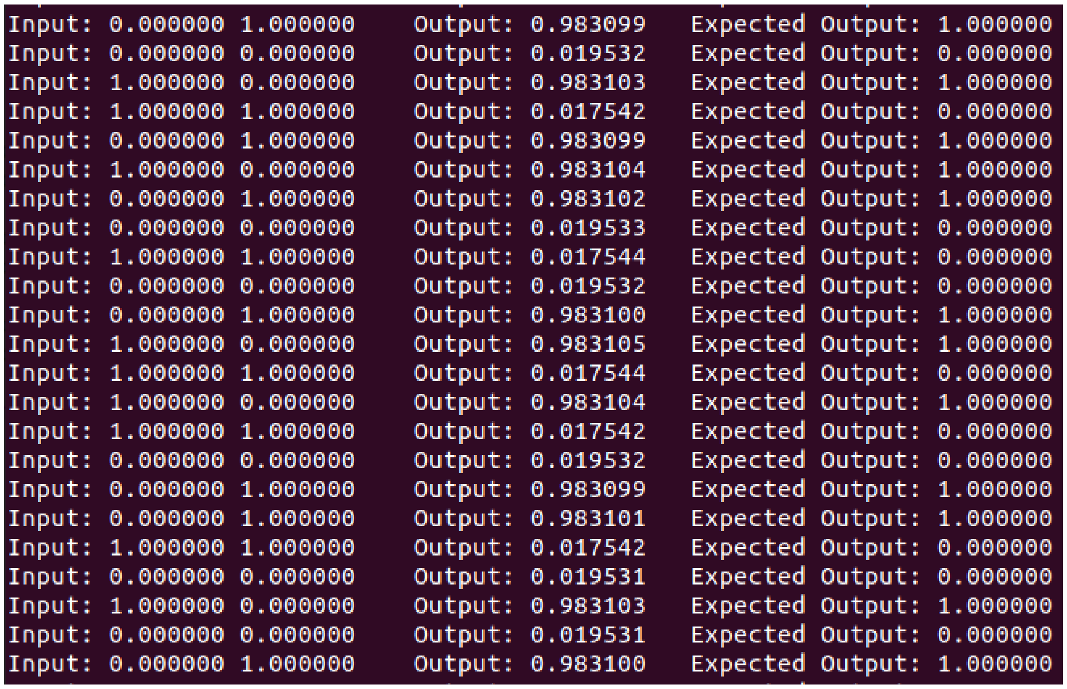


Figure 13: XOR Network

By iterating over a given number of epochs, this XOR Network can minimize its errors and make it learn the XOR operation efficiently.

Here is an example of the algorithm going through many different epochs.

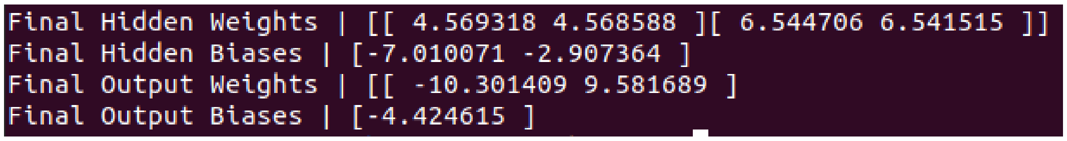


```
Input: 0.000000 1.000000    Output: 0.983099   Expected Output: 1.000000
Input: 0.000000 0.000000    Output: 0.019532   Expected Output: 0.000000
Input: 1.000000 0.000000    Output: 0.983103   Expected Output: 1.000000
Input: 1.000000 1.000000    Output: 0.017542   Expected Output: 0.000000
Input: 0.000000 1.000000    Output: 0.983099   Expected Output: 1.000000
Input: 1.000000 0.000000    Output: 0.983104   Expected Output: 1.000000
Input: 0.000000 1.000000    Output: 0.983102   Expected Output: 1.000000
Input: 0.000000 0.000000    Output: 0.019533   Expected Output: 0.000000
Input: 1.000000 1.000000    Output: 0.017544   Expected Output: 0.000000
Input: 0.000000 0.000000    Output: 0.019532   Expected Output: 0.000000
Input: 0.000000 1.000000    Output: 0.983100   Expected Output: 1.000000
Input: 1.000000 0.000000    Output: 0.983105   Expected Output: 1.000000
Input: 1.000000 1.000000    Output: 0.017544   Expected Output: 0.000000
Input: 1.000000 0.000000    Output: 0.983104   Expected Output: 1.000000
Input: 1.000000 1.000000    Output: 0.017542   Expected Output: 0.000000
Input: 0.000000 0.000000    Output: 0.019532   Expected Output: 0.000000
Input: 0.000000 1.000000    Output: 0.983099   Expected Output: 1.000000
Input: 0.000000 1.000000    Output: 0.983101   Expected Output: 1.000000
Input: 1.000000 1.000000    Output: 0.017542   Expected Output: 0.000000
Input: 0.000000 0.000000    Output: 0.019531   Expected Output: 0.000000
Input: 1.000000 0.000000    Output: 0.983103   Expected Output: 1.000000
Input: 0.000000 0.000000    Output: 0.019531   Expected Output: 0.000000
Input: 0.000000 1.000000    Output: 0.983100   Expected Output: 1.000000
```

Figure 14: XOR Network going through epochs

Overtime, the output slowly gets closer to the expected output. By comparing the obtained output with the expected one, the algorithm is able to get better results.

Finally, the final weights and biases are printed on the screen



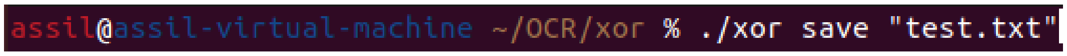
```
Final Hidden Weights | [[ 4.569318 4.568588 ][ 6.544706 6.541515 ]]
Final Hidden Biases | [-7.010071 -2.907364 ]
Final Output Weights | [[ -10.301409 9.581689 ]
Final Output Biases | [-4.424615 ]
```

Figure 15: Final Results

7.2 Save and Load

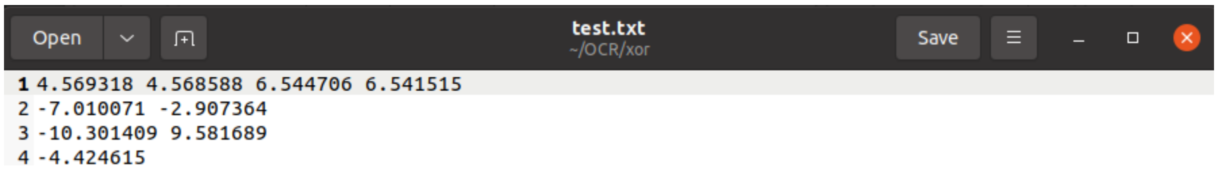
The **XOR Network** can also save the obtained **weights** and **biases** after going through the algorithm.

By using the ”**save**” keyword, the final results of the algorithm can be saved into a specified path. (Can be non-exist).



```
assil@assil-virtual-machine ~/OCR/xor % ./xor save "test.txt"
```

Figure 16: Starting the XOR Network and saving



```
4.569318 4.568588 6.544706 6.541515
-7.010071 -2.907364
-10.301409 9.581689
-4.424615
```

Figure 17: Weights and Biases saved into the test.txt file

The same process can be done with the load keyword. The **XOR Network** can start with a specific set of weights and biases instead of randomly generated ones.

8 Conclusion

Thanks to the great team cohesion and individual work, the project's advancement is going well. The deadlines have been respected and the current state of the project is promising for the future. The project should be finished by the end of the semester.