OCR - First Report

by MAKA-RENA

09 November 2022

Contents

1 The Team **3**
1.1 Armand BLIN . . . 3
1.2 Khaled MILI . . . 4
1.3 Maxim BOCQUILLION . . . 5
1.4 Aurélien DAUDIN . . . 6

2 Task Management **7**

3 Image Processing **8**
3.1 Grayscale Filter . . . 8
3.2 Gaussian Blur . . . 9
3.3 Contrast ameliorations . . . 9
3.4 Black and White . . . 10
3.5 Sobel . . . 10
3.6 Flood filling . . . 11
3.7 Problems encountered . . . 13
3.7.1 First defense . . . 13
3.7.2 Second defense . . . 13
3.8 Manual Rotation . . . 14

4 Grid Detection **15**
4.1 Line Detection (Hough transform) . . . 16
4.2 Automatic Rotation . . . 17

5 Grid Splitting **18**
5.1 Intersections . . . 18
5.2 Crop Function . . . 19

6 Neural Network **21**
6.1 XOR Neural Network . . . 21
6.1.1 Construction Steps . . . 21
6.1.2 Implemented Libraries . . . 21
6.1.3 Main Procedure . . . 22
6.2 Character Recognition Neural Network . . . 23
6.2.1 Neural Network Structure . . . 23
6.2.2 Save and Load . . . 24
6.2.3 Training Data . . . 24
6.2.4 Success Rate . . . 25
6.2.5 DataSet Generation . . . 26
6.2.6 DataSet Generation . . . 26
6.3 Final Steps . . . 27

7 Parser, Solver and Display **28**
7.1 Parser . . . 28
7.2 Sudoku validity check . . . 28

7.3 Solver 28
7.4 Result display 29

8 Website API **30**
8.1 Webiste 30
8.1.1 Homepage 30
8.1.2 Upload picture 30
8.1.3 Result section 31
8.1.4 Team section 31
8.2 API 32

9 Global process of our OCR **33**

10 Conclusion **33**

1 The Team

1.1 Armand BLIN

Hello, my name is Armand, I am very happy to be the project leader. This project allows us to surpass ourselves. More than for the S2 project in my opinion. Which is great, because it gives us even more of a taste of our future job. I am very pleased with what we were able to accomplish in this first working session. I can't wait to show you our final product for the second defense.

Figure 1: Armand BLIN

1.2 Khaled MILI

The explanation behind any phenomenon is what always has amazed me and animated me. It felt like a dream-come-true when I heard about the OCR project. In fact, until this project, I felt like I have never contributed to something meaningful in a scientific point of view. That is why I see this project as the consecration of many years of studies. At a first glimpse, the project seemed difficult to begin. Giving our current skills in programming, I thought that it would be impossible to make: especially in a low-level language as C. Fortunately, I am very pleased to show you what we have accomplished until now. The best is yet to come.

Figure 2: Khaled MILI

1.3 Maxim BOCQUILLION

Growing up, I fell in love with science. The process of looking for answers to unsolved problem and the satisfaction to find the answers (sometimes) was what was driving me. I ended up at Epita without any skills in programming. I never thought that after one year of programming I would be able to take place in a project like this one. It brings important knowledge whether it is in small project management, problem solving or in computer programming.

Figure 3: Maxim BOCQUILLION

1.4 Aurélien DAUDIN

I always had a passion to solve problems. Physic and Maths were amazing for this. But it was only theoretical. I discovered the problem solving in IT in high school. It was really pleasant. You directly see the result and there is a lot of logic. The best part was to apply maths to every days problems. EPITA's projects feel like professional problems. We have to work in team and to schedule and dispatch our work. It is very pleasant and give us a lot of responsibilities and experience. Moreover, OCR is a project base on mathematical formulas and protocols and this is really the part that I like. I learned a lot about the C language, image analysis and memory management.

Figure 4: Aurélien DAUDIN

2 Task Management

Below is the distribution of tasks for this first defense.

Task - Name	Armand	Khaled	Maxim	Aurélien	Last Defense
Image processing					
Image Loading			Principal	Second	100%
Color Removing			Principal	Second	100%
Filter the Image			Principal	Principal	100%
Rotation	Principal			Second	100%
Grid detection			Second	Principal	100%
Image resizing			Principal	Principal	100%
Digit recognition					
Neural Network		Principal			100%
AI Dataset		Principal			100%
Other					
Sudoku solving	Principal				100%
Graphic Interface	Principal				100%
Web site	Principal				100%

Table 1: Task management tab

3 Image Processing

In order to analyze the sudoku in deep, we first have to do some Image Processing. It will help the computing process to detect the lines of a sudoku and the numbers. To do so, we had to implement multiple actions.

3.1 Grayscale Filter

The first action implemented is the grayscale action. The process is simple : it consists in turning a colorful image into an image using only shades of grey. To convert an image in grayscale, we traverse every pixels of the image. We save the values of red, blue, and grey corresponding to the color of the image, and make an average of those using the formula :

$$p_{(x,y)} = p_{(x,y)_r} * 0.3 + p_{(x,y)_g} * 0.59 + p_{(x,y)_b} * 0.11$$

$p_{(x,y)}$: pixel value $\in [0, 255]$
$_r$: red color $_g$: grey color $_b$: blue color

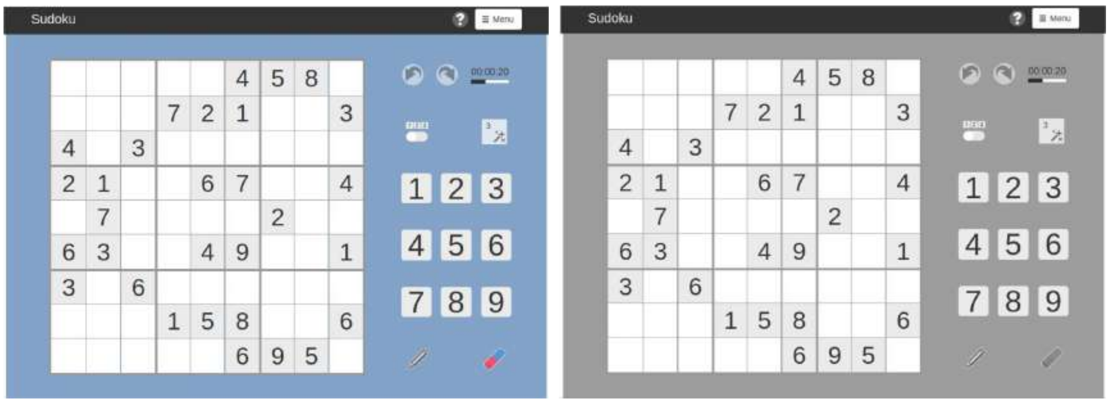


Figure 5: Before vs After Grayscale Filter

3.2 Gaussian Blur

Once that our picture is grayscaled, we apply the Gaussian blur. The Gaussian blur consists in blurring the image using kernel convolution to reduce the noise of an image But what is kernel convolution ?
A Kernel convolution is a process where we take small grids of number, here 3*3, and we pass them over the whole picture.

To do so, we go over each pixel of the pictures, we look at the pixel value of the 8 closest neighbors of the pixel we are looking at, but also the value of the latter. We then proceed to add those value that where multiplied by the value at their position in the Gaussian kernel. We then divide the sum by the number of neighbors +1 corresponding to the center value, we thus obtain the new color of the pixel.

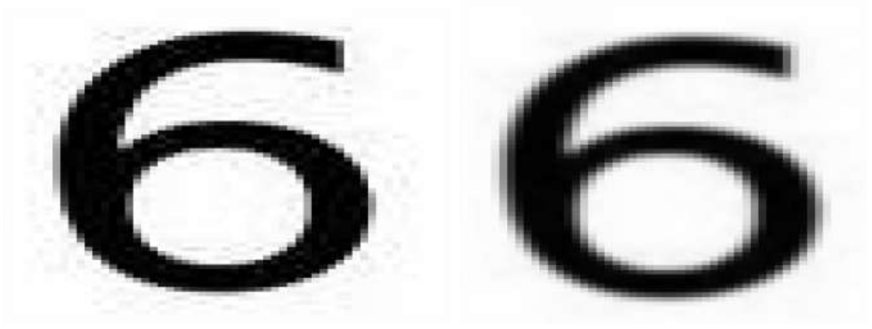


Figure 6: Before vs After Gaussian Filter

3.3 Contrast ameliorations

In order to treat images with low lightning we decided to implement a contrast amelioration function. The principle is simple. It iterates over each pixels of the image and add 10 percents of the value to the pixel.

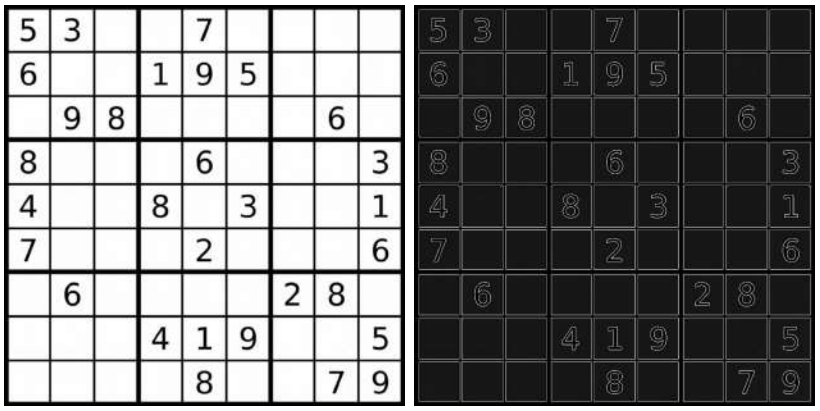


Figure 7: Before vs After Contrast Filter

3.4 Black and White

The black and white treatment turns the image into a black-and-white picture. It goes through each pixel. If the pixel is above a certain threshold that is adjusted for each image, the pixel becomes white, or else it becomes black. This process is a preparation for the Sobel method.

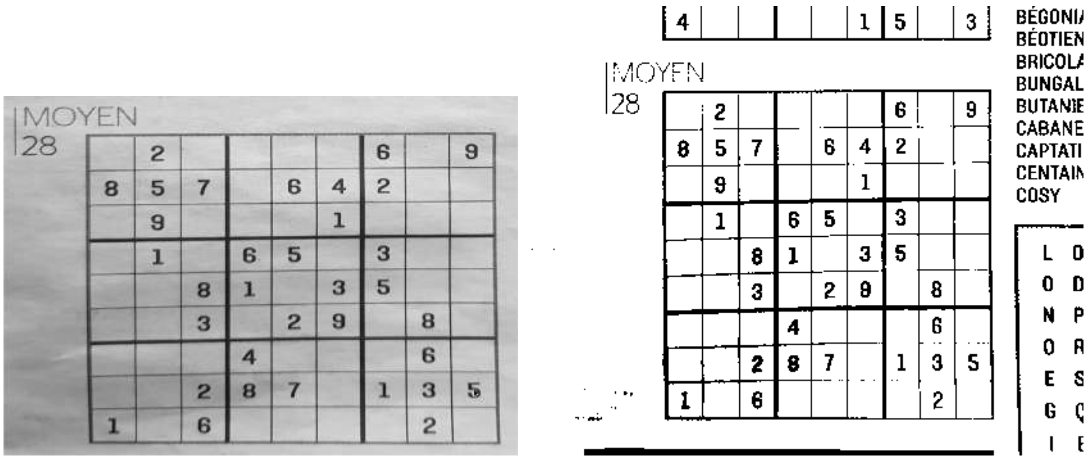


Figure 8: Before vs After Black and White Filter

3.5 Sobel

After applying the grayscale and the Gaussian blur on the sudoku image we make a Sobel treatment. To do so, we use the same principle as for the Gaussian blur. We take the eight neighbors of the pixel x and multiply them with their corresponding position in the two kernels. We add every x and y thus obtain. The formula : is

$G = \sqrt{G(x)^2 + G(y)^2}$

then used corresponding to define the gradient of the image.

$$\begin{pmatrix} -1.0 & 0.0 & 1.0 \\ -2.0 & 0.0 & 2.0 \\ -1.0 & 0.0 & 1.0 \end{pmatrix}$$

Figure 9: X Kernel for Sobel

$$\begin{pmatrix} -1.0 & -2.0 & -1.0 \\ 0.0 & 0.0 & 0.0 \\ 1.0 & 2.0 & 1.0 \end{pmatrix}$$

Figure 10: Y Kernel for Sobel

These gradient values will tell us where are the brutal color variations. By rescaling these values on 0 to 255 we obtain white lines that represents the border of the picture.
Then we need to process these gradient values. We analyse them and thanks to the second data, the angle computed with the formula O = atan2(Gy,Gx), we can filter these data to obtain only chains (closed paths) that will represent the borders of forms in the pictures. By applying a second filter we obtain a clear and define picture like the one bellow

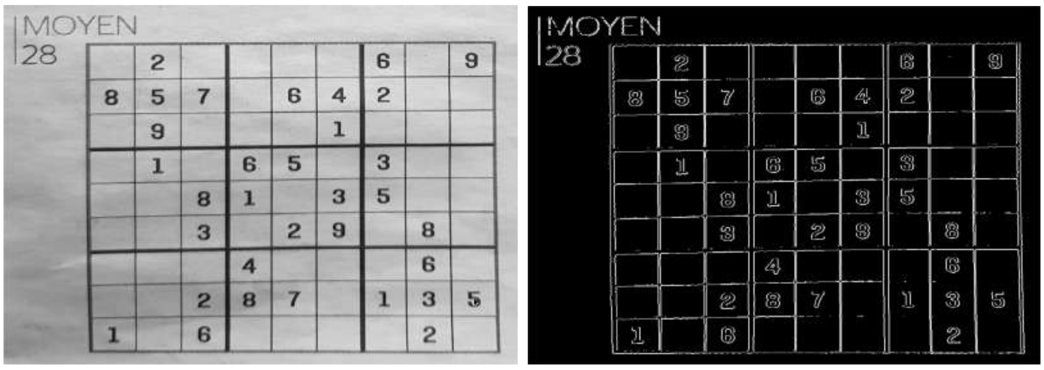


Figure 11: Before vs After Sobel Filter

3.6 Flood filling

After applying the sobel filter, the goal is to find the sudoku grid. In order to do that, we apply the flood filling principal. We iterate over each pixel. If the pixel is white, we look if the neighbors (up, down, left and right) of this pixel are white as well recursively. We stop when there is no white pixel in the area. We keep the highest number of pixel that are linked together, representing usually the grid and draw them in a specific color, here red. We then delete everything that is not included in that red grid in order to only keep what is important for us: the sudoku grid and its numbers.

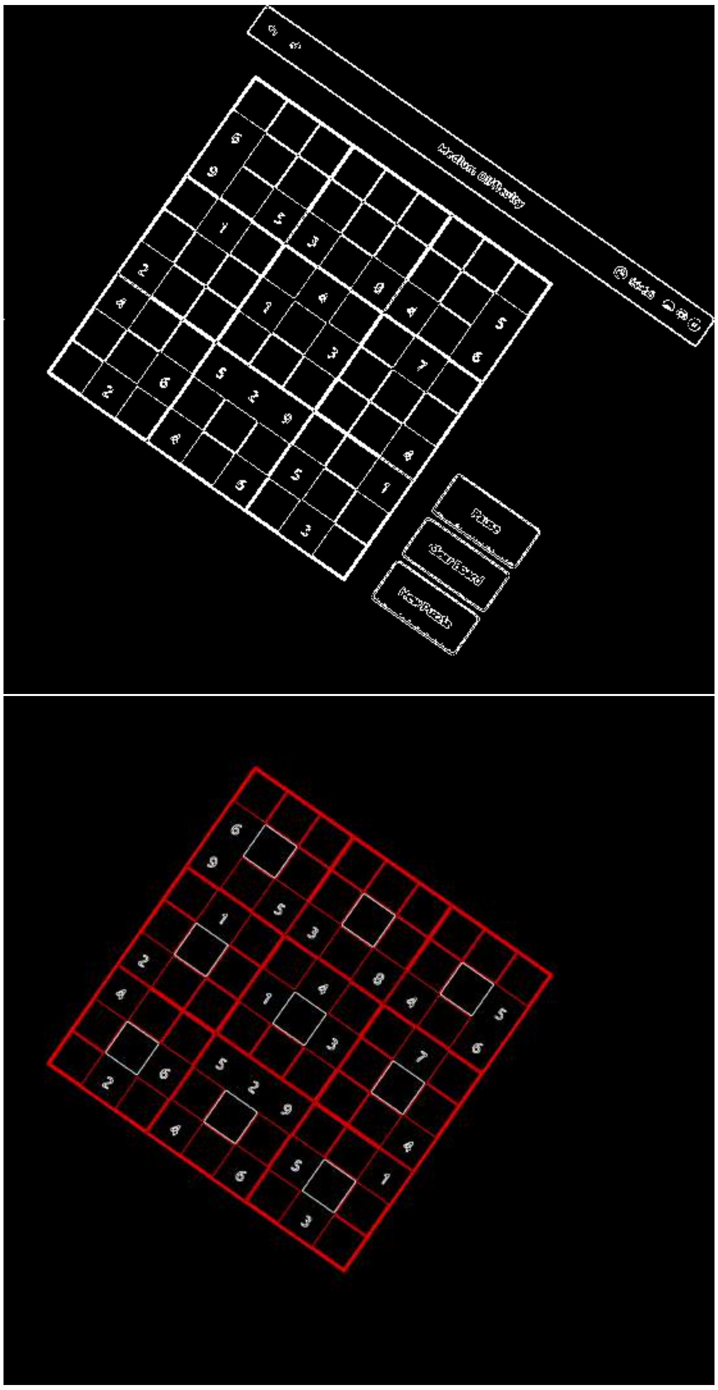


Figure 12: Before vs After Flood Filling

3.7 Problems encountered

3.7.1 First defense

- When we first did our algorithms for Sobel and Gaussian, the drawn pictures were all black. This was mainly due to the type of variables contained in the arrays used to draw our new pictures. We thus decided to use Uint32.

- In the first version of Gaussian, we had segmentation fault on some pictures we tested. We decided to move forward with the sobel implementation. We had even more segfaults problems with pictures that did not have this problem at first. To fix those problems, we sought for multiple debugging programs such as Valgrind and GDB. Gdb was the one that got us out of problem. It indicated that we did not allow enough -memory space when initializing arrays. Thus, we decided to use the malloc function.

3.7.2 Second defense

- We add to improve our color treatment because it was not precise enough at the first defense causing some problems in the detection of the lines and the grid.

3.8 Manual Rotation

An important part of this OCR is the rotation of an image. It is needed to analyse and detect the grid. We started with a change of pixel location from an image A to an image B. At the beginning the image B is a totally blank image to which we add to it progressively pixels of colors selected in the image A. That allowed us not to have two superimposed images. Moreover, this rotation works thanks to an angle θ. This will allow to make an "automatic rotation" when detecting the angle between the sudoku horizontal line closest to the bottom of the image.

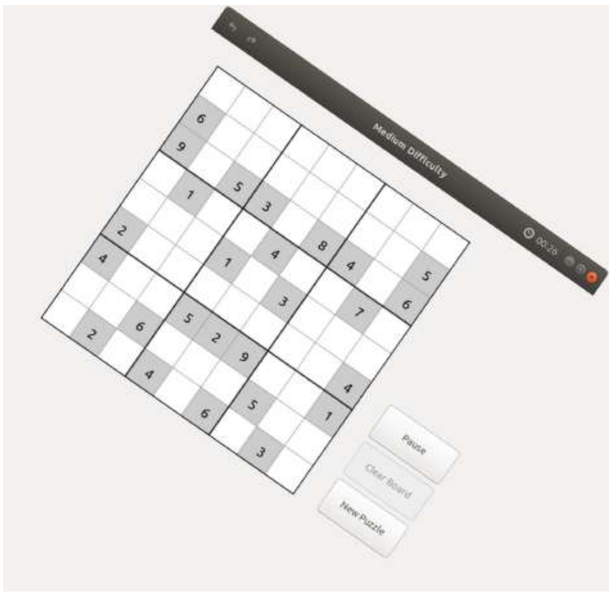


Figure 13: Before Manual Rotation

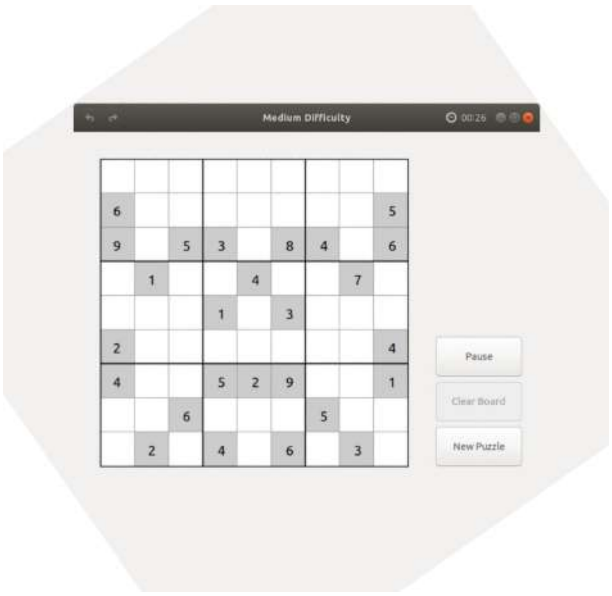


Figure 14: After Manual Rotation with angle $\theta = 35$

This is an example of rotation code:

```
x = (row - new_w / 2) * cos_angle
            - (col - new_h / 2) * sin_angle + w / 2;
y = (row - new_w / 2) * sin_angle
            + (col - new_h / 2) * cos_angle + h / 2;

if (x >= 0 && x < w && y >= 0 && y < h)
{
    SDL_GetRGB(pixels[y * w + x], image->format, &r, &g, &b);
    new_pixels[col * new_w + row]=SDL_MapRGB(rotated->format, r, g, b);
}
```

Figure 15: Extract of the code of rotation

4 Grid Detection

We need to analyse the image to detect the grid. Indeed, the picture can have other elements such as text or a color background. The grid could be a little part of the entire picture.
As presented before, we have the Flood fill algorithm in the process that detect the grid. However, the grid is also composed of lines and digits. Thus, we need to detect these lines and also determine if our image is straight so that we can then read the cells correctly.

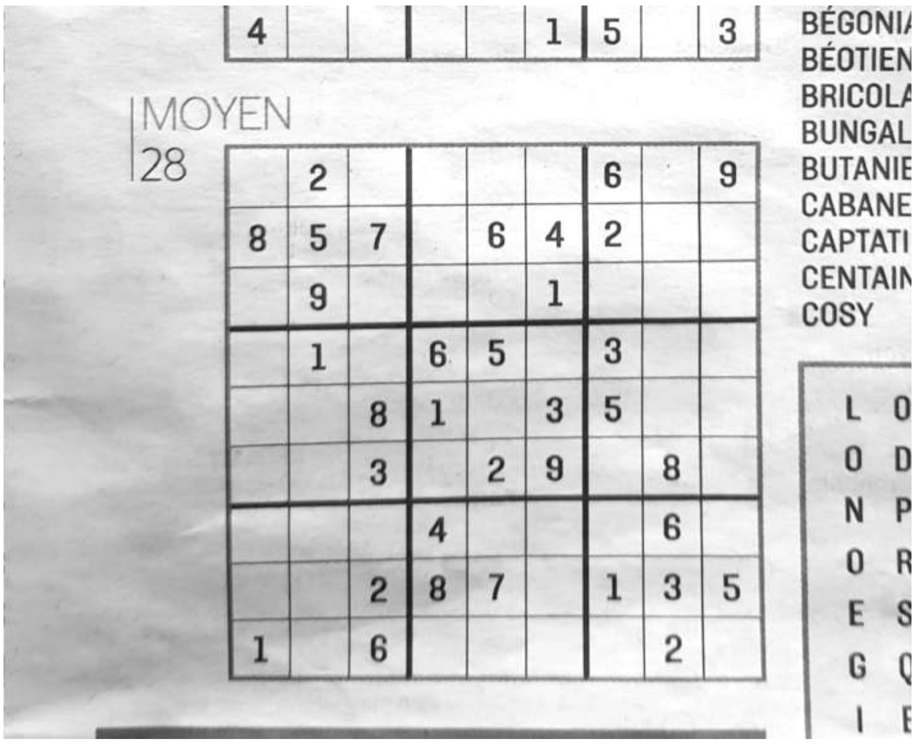


Figure 16: Image whith perturbations

4.1 Line Detection (Hough transform)

Hough transform is a featured extraction technique used to find the lines in an image for example by processing it. We will use this method to detect the grid and the cells. It will be the first step of the line detection.

This process will return the lines found on the picture. Here is the result of this process on the first picture.

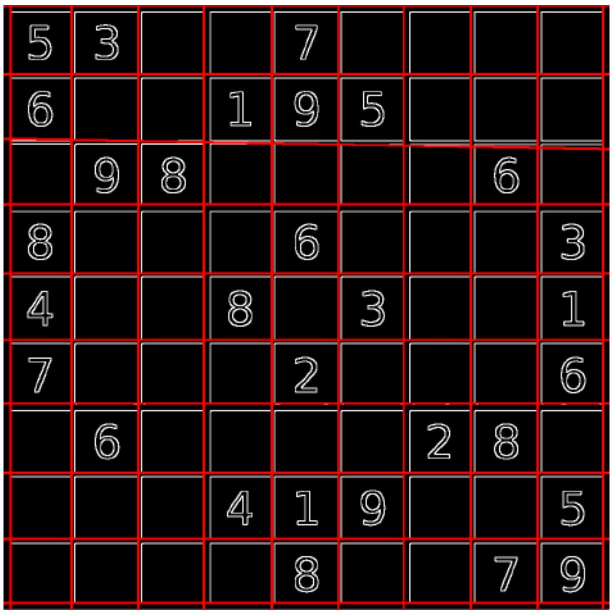


Figure 17: Image representing in red all the lines in picture 1

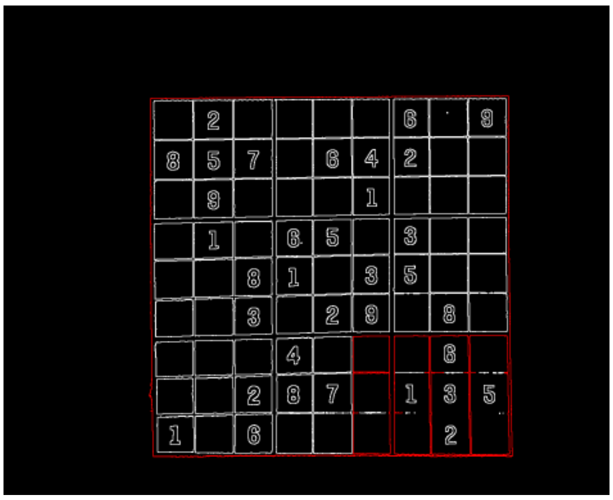


Figure 18: Image representing in red all the lines in picture 2

4.2 Automatic Rotation

Hough Transform algorithm returns a pair of value rho and theta for each line.
These two values are very useful to analyse the image. By using the lines it is possible to find the general picture orientation. To do so we will compare for each line their orientation and keep the maximum.
The orientation is given by the theta as it can be seen on the picture bellow for the first line of the grid.

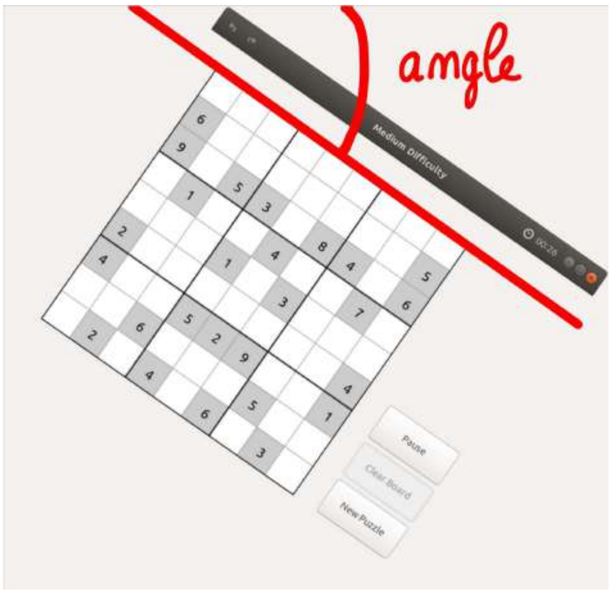


Figure 19: Angle theta of the first line

Then a call to the manual rotation function is done with this value. The result is a perfectly straight picture.

5 Grid Splitting

After the grid detection, we need to split it into smaller images. These smaller images will be the cells that we will analyse with the Neural Network.

The difficulty is to detect precisely the cells even when the picture is not straight. Moreover, we need perfectly clean pictures with just the number in it to increase the performances of the neural network.
The Flood fill algorithm detect the grid and gives us coordinates to use as bounds. By adding the Hough transform algorithm and the automatic rotation we obtain a perfect straight square representing our grid.

The challenge is now to split this grid precisely. The goal is to get rid of everything except numbers.

5.1 Intersections

The Hough transform algorithm returns for each lines an equation in polar coordinates. To simplify the intersections computations, these polar coordinates are turned into Cartesian coordinates.
Then by solving the equation between two lines we obtain their intersection points. It is a basic maths problem to solve but here the difficulty is to consider each combinations between the lines.

After this function we obtain a list of coordinates representing the intersections. The result can be seen on the picture bellow.

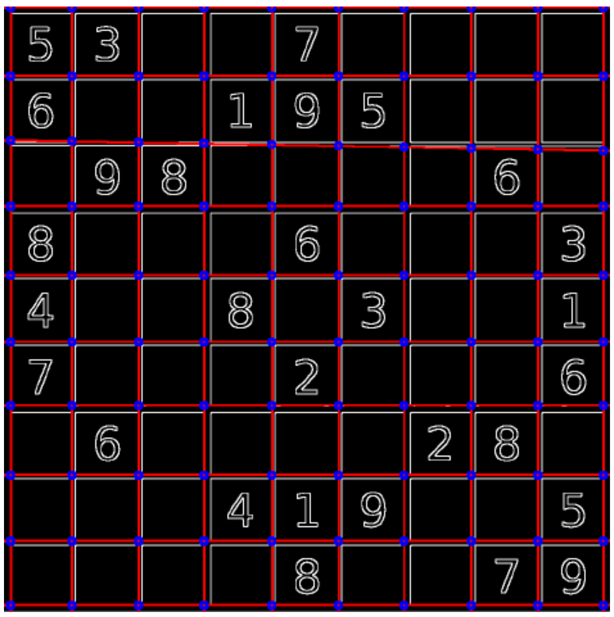


Figure 20: Image representing in blue all the intersections in picture 1

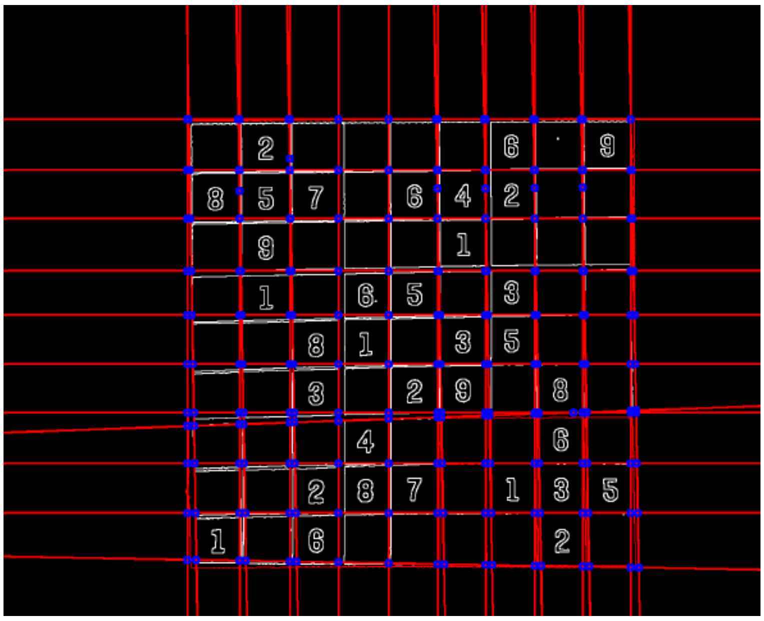


Figure 21: Image representing in blue all the intersections in picture 2

5.2 Crop Function

The crop function takes as parameter the principal image and two coordinates corresponding to the top left corner and the bottom right one.
The principal difficulty is to initialize the new blank image to the good size before copying the wanted pixels. Because the matrix representing the picture is in major-row implementation, we needed to find a formula to obtain the width and the height of the cropped picture.
This function is called by the one finding the coordinates of the cells. This will give us 81 pictures named from 0 to 80 representing each cell. This numerical name allow us to associate each picture to the corresponding place in the grid.

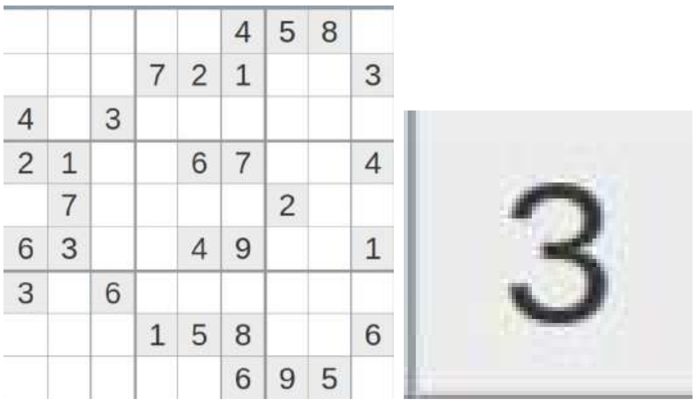


Figure 22: Before vs After Resize

After using the crop process, we need to apply a function in order to resize the image into a 28 by 28 pixel image for the neural network.

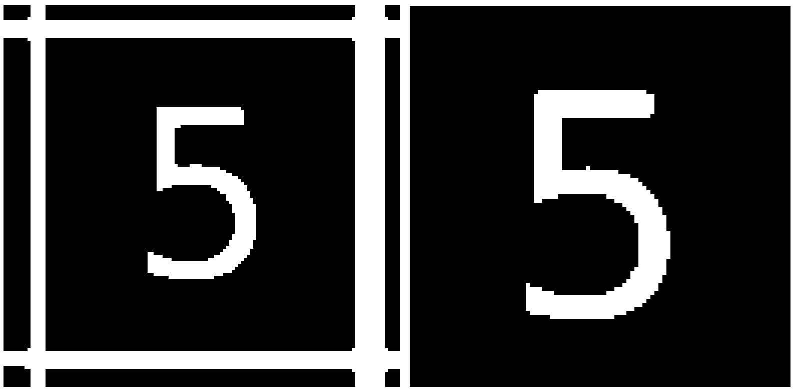


Figure 23: Before vs After Cleaning

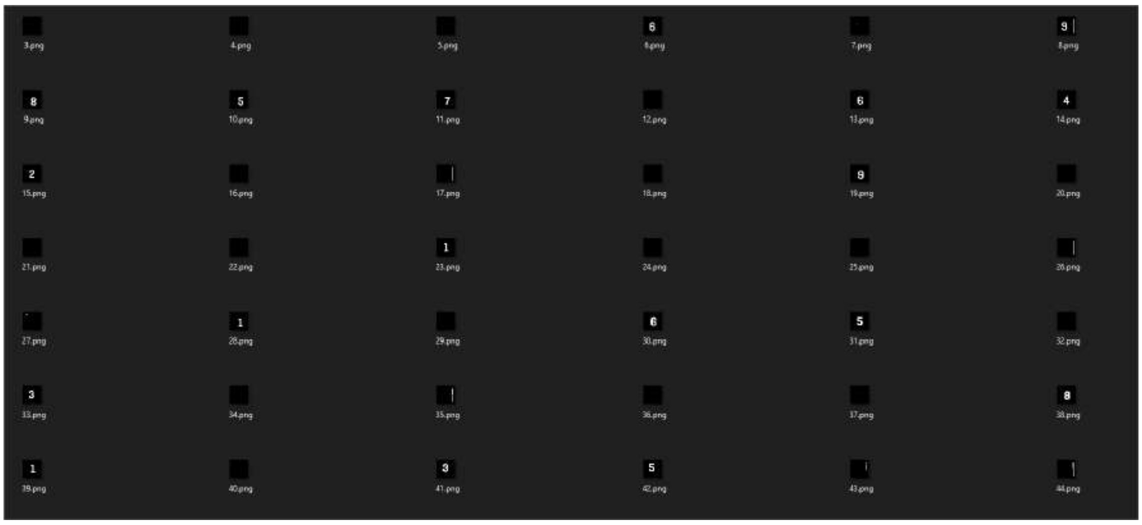


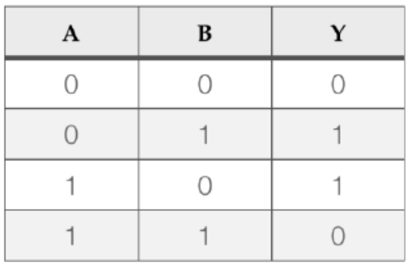
Figure 24: All the cells numbered pictures

6 Neural Network

In order to make a neural network that is able to recognize characters (handwritten or not) we had to start with something easy: a proof of concept. We were asked to make a neural network that is able to learn the XOR gate.

6.1 XOR Neural Network

Our neural network is composed of a **forward propagation**, and a **backward propagation** using the gradient descent algorithm. The aim was to have a predicted output that converges to the actual values of the XOR gate.



A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

Figure 25: Xor Truth Table

6.1.1 Construction Steps

The hardest part was without any doubt to understand how a neural network works. For that reason, we decided to search for different resources in order to find a correct approach to start our neural network.

6.1.2 Implemented Libraries

Our first observation was the number of matrix calculations that a neural network must do to train. Thus, it was almost necessary for us to implement a matrix library to facilitate our work. We preferred a matrix structure to work with to the basic C arrays. It seemed visually more appealing for us and even more clever to understand. In fact, we made this implementation in a manner so we can easily detect and fix the different problems encountered. Basically, we implemented the important and common operations (addition, subtraction, dot-product...), and the attributes ”rows” and ”cols” to easily access the size of the matrices.

6.1.3 Main Procedure

Once the needed tools implemented, we started working on our neural network. We created one that consists of one input layer (containing two nodes), a hidden layer (with two nodes), and an output layer with one node.

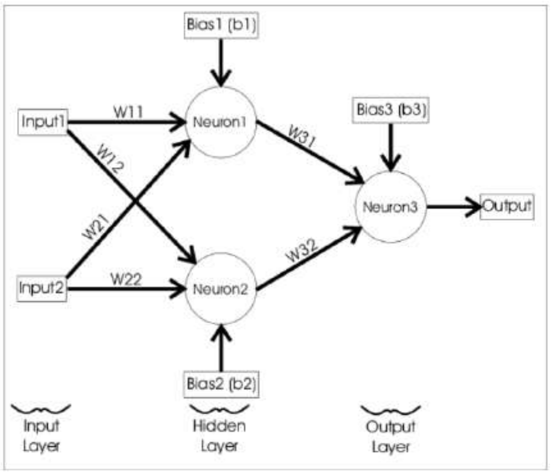
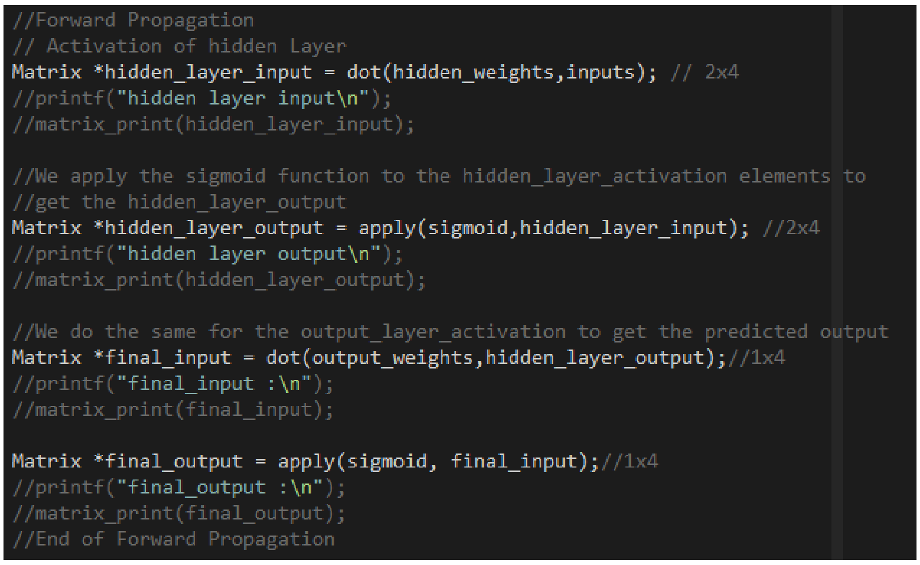


Figure 26: Graphical Representation of Xor Neural Network

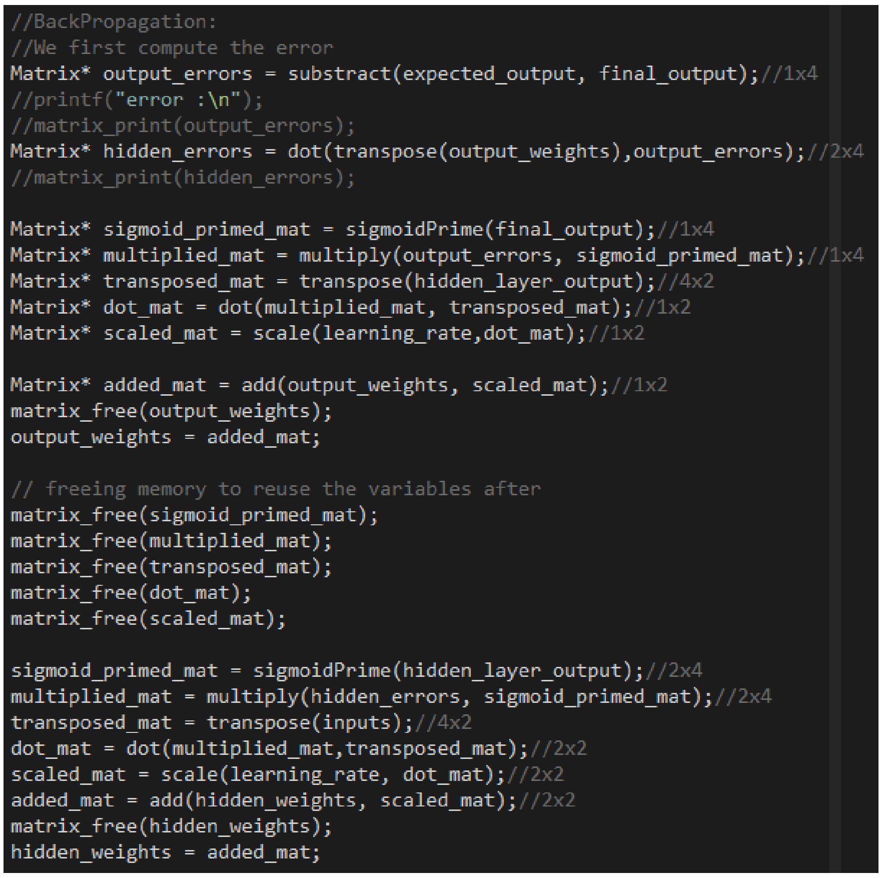
First of all, we initialize our hidden and output weights with random values, using a uniform distribution(that we implemented in the matrix library). After that, we start the forward pass (or forward propagation). Hence, at each iteration, we compute the new values of the predicted output, considering the change of parameters in the weights that happens in the backward propagation.



```
//Forward Propagation
// Activation of hidden Layer
Matrix *hidden_layer_input = dot(hidden_weights,inputs); // 2x4
//printf("hidden layer input\n");
//matrix_print(hidden_layer_input);

//We apply the sigmoid function to the hidden_layer_activation elements to
//get the hidden_layer_output
Matrix *hidden_layer_output = apply(sigmoid,hidden_layer_input); //2x4
//printf("hidden layer output\n");
//matrix_print(hidden_layer_output);

//We do the same for the output_layer_activation to get the predicted output
Matrix *final_input = dot(output_weights,hidden_layer_output);//1x4
//printf("final_input :\n");
//matrix_print(final_input);

Matrix *final_output = apply(sigmoid, final_input);//1x4
//printf("final_output :\n");
//matrix_print(final_output);
//End of Forward Propagation
```

Figure 27: Implementation of the Forward Propagation

The idea behind the backward propagation is to compute the error between the predicted output and the expected one (meaning the actual output that the Xor operation returns) and to adjust the weights according to the error values. That is precisely what is done in this part of code:



```
//BackPropagation:
//We first compute the error
Matrix* output_errors = substract(expected_output, final_output);//1x4
//printf("error :\n");
//matrix_print(output_errors);
Matrix* hidden_errors = dot(transpose(output_weights),output_errors);//2x4
//matrix_print(hidden_errors);

Matrix* sigmoid_primed_mat = sigmoidPrime(final_output);//1x4
Matrix* multiplied_mat = multiply(output_errors, sigmoid_primed_mat);//1x4
Matrix* transposed_mat = transpose(hidden_layer_output);//4x2
Matrix* dot_mat = dot(multiplied_mat, transposed_mat);//1x2
Matrix* scaled_mat = scale(learning_rate,dot_mat);//1x2

Matrix* added_mat = add(output_weights, scaled_mat);//1x2
matrix_free(output_weights);
output_weights = added_mat;

// freeing memory to reuse the variables after
matrix_free(sigmoid_primed_mat);
matrix_free(multiplied_mat);
matrix_free(transposed_mat);
matrix_free(dot_mat);
matrix_free(scaled_mat);

sigmoid_primed_mat = sigmoidPrime(hidden_layer_output);//2x4
multiplied_mat = multiply(hidden_errors, sigmoid_primed_mat);//2x4
transposed_mat = transpose(inputs);//4x2
dot_mat = dot(multiplied_mat,transposed_mat);//2x2
scaled_mat = scale(learning_rate, dot_mat);//2x2
added_mat = add(hidden_weights, scaled_mat);//2x2
matrix_free(hidden_weights);
hidden_weights = added_mat;
```

Figure 28: Implementation of the Backward Propagation

6.2 Character Recognition Neural Network

Once the Xor neural network was finished, we started the implementation of the final neural network: the one that will recognize the different characters. We basically used the same structure as the one used to train the Xor network. Except, this time we needed to find a way to save the data on the weights after the training part and to load it once we wanted to use our neural network. We also needed to find a more compact way to manipulate the network data in the different calculations.

6.2.1 Neural Network Structure

For all those reasons we decided to create a neural network structure to centralize all the data that we needed. To tell the truth, it helped us considerably to manage the different parts of our code, precisely when we needed to save a neural network and to load it afterwards.

6.2.2 Save and Load

We simply had to write a function that takes the network and a path in parameters and that creates a folder containing a descriptor file with the information about the network (the layers), and two files, one with the saved hidden weights and one with the saved output weights.
To load a network, we wrote a function that takes the path of a folder as a parameter and extracts the data contained in the different files to associate them with the attributes of the neural network.

6.2.3 Training Data

We needed a set of data to train our neural network. We decided to take the MNIST data set (a dataset containing images of handwritten digits). The only problem was that it was either written in binary files or in ".csv". Thus, we wrote a function to translate the csv-files into our matrix implementation. That way, we can now train our neural network with the correct format of data and without any problem of calculation.
If we focus well, we can observe that the number 4 for instance appears inside the matrix below while using the translating function.

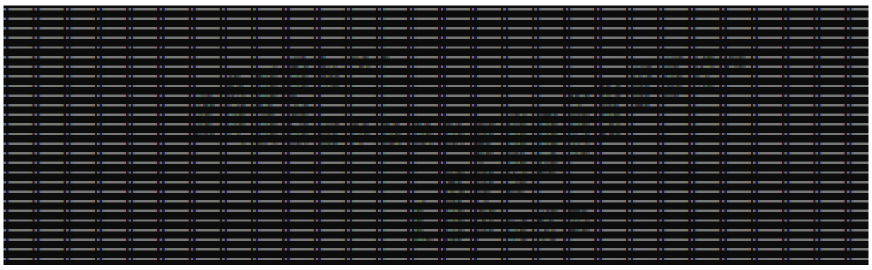
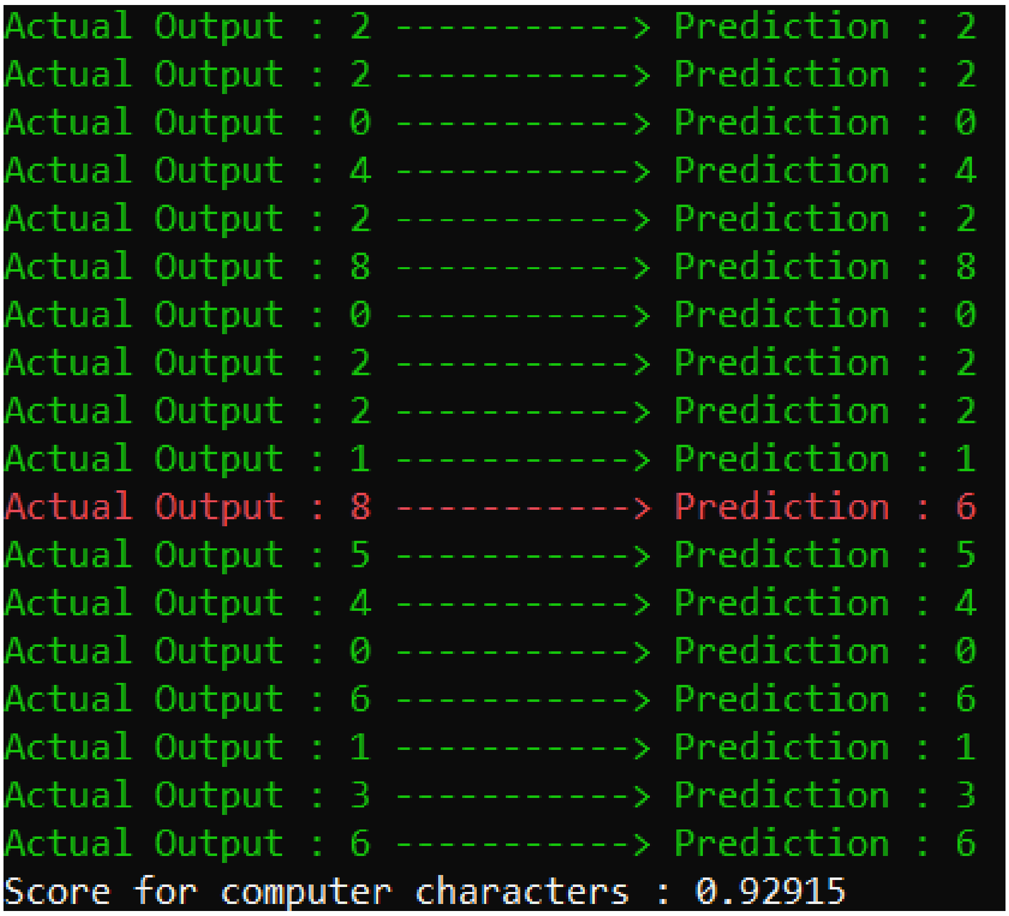


Figure 29: Result of Digit Conversion from Data-file to Matrix

6.2.4 Success Rate

Through all these steps, we managed to have a neural network fully functional and able to recognize handwritten characters with a succes rate of 94 percent. Though, when we tested the neural network on computer characters, the result was not that good. Thus, we understood that we needed to use another dataset to make it work.

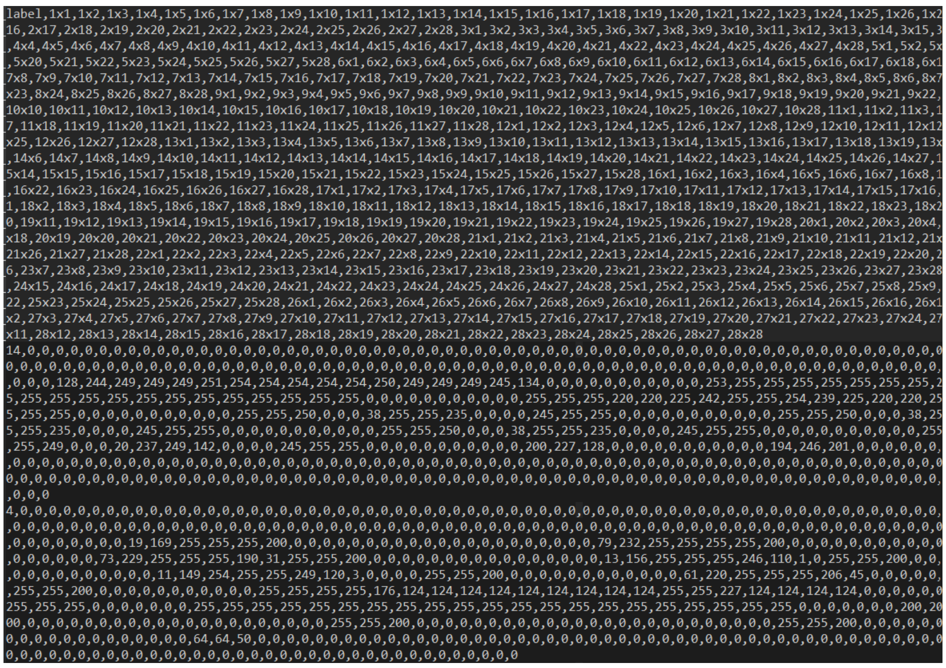


```
Actual Output : 2 -----------> Prediction : 2
Actual Output : 2 -----------> Prediction : 2
Actual Output : 0 -----------> Prediction : 0
Actual Output : 4 -----------> Prediction : 4
Actual Output : 2 -----------> Prediction : 2
Actual Output : 8 -----------> Prediction : 8
Actual Output : 0 -----------> Prediction : 0
Actual Output : 2 -----------> Prediction : 2
Actual Output : 2 -----------> Prediction : 2
Actual Output : 1 -----------> Prediction : 1
Actual Output : 8 -----------> Prediction : 6
Actual Output : 5 -----------> Prediction : 5
Actual Output : 4 -----------> Prediction : 4
Actual Output : 0 -----------> Prediction : 0
Actual Output : 6 -----------> Prediction : 6
Actual Output : 1 -----------> Prediction : 1
Actual Output : 3 -----------> Prediction : 3
Actual Output : 6 -----------> Prediction : 6
Score for computer characters : 0.92915
```

Figure 30: Succes Rate Display of the Computer Character Prediction

6.2.5 DataSet Generation

Eventually, as the MNIST Dataset was not accurate enough for our network to recognize computer characters. In order to solve this problem, we had to generate our own dataset. Thus, we used the pillow library of python to do so. We also created a testing dataset with the same library. Obviously, we used completely different images in the two datasets to see if our network was training well.



```
label,1x1,1x2,1x3,1x4,1x5,1x6,1x7,1x8,1x9,1x10,1x11,1x12,1x13,1x14,1x15,1x16,1x17,1x18,1x19,1x20,1x21,1x22,1x23,1x24,1x25,1x26,1x2
16,2x17,2x18,2x19,2x20,2x21,2x22,2x23,2x24,2x25,2x26,2x27,2x28,3x1,3x2,3x3,3x4,3x5,3x6,3x7,3x8,3x9,3x10,3x11,3x12,3x13,3x14,3x15,3
,4x4,4x5,4x6,4x7,4x8,4x9,4x10,4x11,4x12,4x13,4x14,4x15,4x16,4x17,4x18,4x19,4x20,4x21,4x22,4x23,4x24,4x25,4x26,4x27,4x28,5x1,5x2,5x
,5x20,5x21,5x22,5x23,5x24,5x25,5x26,5x27,5x28,6x1,6x2,6x3,6x4,6x5,6x6,6x7,6x8,6x9,6x10,6x11,6x12,6x13,6x14,6x15,6x16,6x17,6x18,6x1
7x8,7x9,7x10,7x11,7x12,7x13,7x14,7x15,7x16,7x17,7x18,7x19,7x20,7x21,7x22,7x23,7x24,7x25,7x26,7x27,7x28,8x1,8x2,8x3,8x4,8x5,8x6,8x7
x23,8x24,8x25,8x26,8x27,8x28,9x1,9x2,9x3,9x4,9x5,9x6,9x7,9x8,9x9,9x10,9x11,9x12,9x13,9x14,9x15,9x16,9x17,9x18,9x19,9x20,9x21,9x22,
10x10,10x11,10x12,10x13,10x14,10x15,10x16,10x17,10x18,10x19,10x20,10x21,10x22,10x23,10x24,10x25,10x26,10x27,10x28,11x1,11x2,11x3,1
7,11x18,11x19,11x20,11x21,11x22,11x23,11x24,11x25,11x26,11x27,11x28,12x1,12x2,12x3,12x4,12x5,12x6,12x7,12x8,12x9,12x10,12x11,12x12
x25,12x26,12x27,12x28,13x1,13x2,13x3,13x4,13x5,13x6,13x7,13x8,13x9,13x10,13x11,13x12,13x13,13x14,13x15,13x16,13x17,13x18,13x19,13x
,14x6,14x7,14x8,14x9,14x10,14x11,14x12,14x13,14x14,14x15,14x16,14x17,14x18,14x19,14x20,14x21,14x22,14x23,14x24,14x25,14x26,14x27,1
5x14,15x15,15x16,15x17,15x18,15x19,15x20,15x21,15x22,15x23,15x24,15x25,15x26,15x27,15x28,16x1,16x2,16x3,16x4,16x5,16x6,16x7,16x8,1
,16x22,16x23,16x24,16x25,16x26,16x27,16x28,17x1,17x2,17x3,17x4,17x5,17x6,17x7,17x8,17x9,17x10,17x11,17x12,17x13,17x14,17x15,17x16,
1,18x2,18x3,18x4,18x5,18x6,18x7,18x8,18x9,18x10,18x11,18x12,18x13,18x14,18x15,18x16,18x17,18x18,18x19,18x20,18x21,18x22,18x23,18x2
0,19x11,19x12,19x13,19x14,19x15,19x16,19x17,19x18,19x19,19x20,19x21,19x22,19x23,19x24,19x25,19x26,19x27,19x28,20x1,20x2,20x3,20x4,
x18,20x19,20x20,20x21,20x22,20x23,20x24,20x25,20x26,20x27,20x28,21x1,21x2,21x3,21x4,21x5,21x6,21x7,21x8,21x9,21x10,21x11,21x12,21x
21x26,21x27,21x28,22x1,22x2,22x3,22x4,22x5,22x6,22x7,22x8,22x9,22x10,22x11,22x12,22x13,22x14,22x15,22x16,22x17,22x18,22x19,22x20,2
6,23x7,23x8,23x9,23x10,23x11,23x12,23x13,23x14,23x15,23x16,23x17,23x18,23x19,23x20,23x21,23x22,23x23,23x24,23x25,23x26,23x27,23x28
,24x15,24x16,24x17,24x18,24x19,24x20,24x21,24x22,24x23,24x24,24x25,24x26,24x27,24x28,25x1,25x2,25x3,25x4,25x5,25x6,25x7,25x8,25x9,
22,25x23,25x24,25x25,25x26,25x27,25x28,26x1,26x2,26x3,26x4,26x5,26x6,26x7,26x8,26x9,26x10,26x11,26x12,26x13,26x14,26x15,26x16,26x1
x2,27x3,27x4,27x5,27x6,27x7,27x8,27x9,27x10,27x11,27x12,27x13,27x14,27x15,27x16,27x17,27x18,27x19,27x20,27x21,27x22,27x23,27x24,27
x11,28x12,28x13,28x14,28x15,28x16,28x17,28x18,28x19,28x20,28x21,28x22,28x23,28x24,28x25,28x26,28x27,28x28
14,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
,0,0,0,128,244,249,249,249,251,254,254,254,254,254,250,249,249,249,245,134,0,0,0,0,0,0,0,0,0,0,0,253,255,255,255,255,255,255,255,2
5,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,0,0,0,0,255,255,255,220,220,225,242,255,255,254,239,225,220,220,25
5,255,255,0,0,0,0,0,0,0,0,0,0,0,255,255,250,0,0,0,38,255,255,235,0,0,0,0,245,255,255,0,0,0,0,0,0,0,0,0,0,255,255,250,0,0,0,38,25
5,255,235,0,0,0,0,245,255,255,0,0,0,0,0,0,0,0,0,0,0,255,255,250,0,0,0,38,255,255,235,0,0,0,0,245,255,255,0,0,0,0,0,0,0,0,0,0,255
,255,249,0,0,0,20,237,249,142,0,0,0,0,245,255,255,0,0,0,0,0,0,0,0,0,0,200,227,128,0,0,0,0,0,0,0,0,0,0,0,194,246,201,0,0,0,0,0,0,
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
,0,0,0
4,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,0,19,169,255,255,255,200,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,79,232,255,255,255,255,200,0,0,0,0,0,0,0,0,0,0
,0,0,0,0,0,0,0,73,229,255,255,255,190,31,255,255,200,0,0,0,0,0,0,0,0,0,0,0,0,0,0,13,156,255,255,255,246,110,1,0,255,255,200,0,0,
,0,0,0,0,0,0,0,0,0,0,11,149,254,255,255,249,120,3,0,0,0,0,255,255,200,0,0,0,0,0,0,0,0,0,0,0,0,61,220,255,255,255,206,45,0,0,0,0,0,
,255,255,200,0,0,0,0,0,0,0,0,0,0,0,255,255,255,255,176,124,124,124,124,124,124,124,124,124,255,255,227,124,124,124,124,0,0,0,0,0,0
255,255,255,0,0,0,0,0,0,0,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,255,0,0,0,0,0,0,0,200,20
00,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,200,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,255,255,200,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,64,64,50,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
```

Figure 31: Compression of data into CSV files

6.2.6 DataSet Generation

Once the dataset was generated, we compressed all the images in a ".csv" file, exactly like the MNIST Dataset, to make it compatible with the neural network. We also did the same process for the testing dataset. At that step precisely, when we trained the network with the new data, the succes rate plummeted to reach 0 percent. After hours of debugging, we realized that we were missing something primordial. It is in fact the shuffling of the dataset. Once we did this, the rate success rate reached the 84 percent. We managed to reach the last 10 percent by adjusting the network parameters (learning rate = 0.075, Neurons in hidden layer = 40).

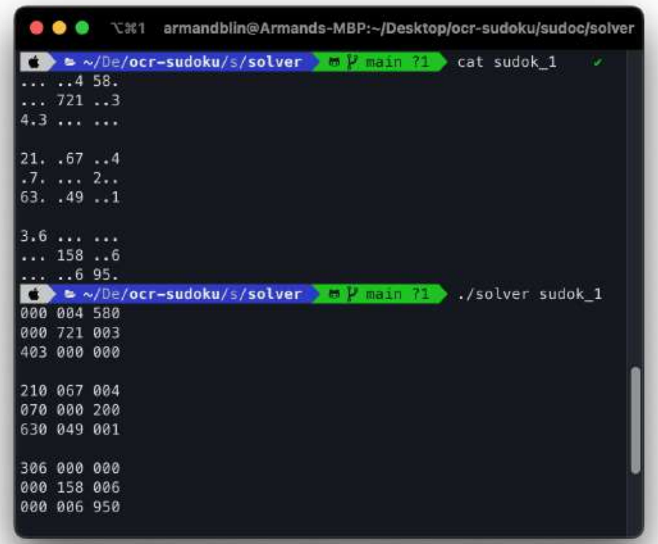
6.3 Final Steps

Eventually, we tried different parameters of hidden neurons, activation functions and learning rates for our network to find the optimal ones. We are pleased to say that we managed to reach a success rate of 94 percent for the computer character recognition, 94 percent for the handwritten character recognition and 81 percent for the hexadecimal character recognition. We implemented the softmax function and the results were pretty much the same as for the sigmoid activation function. Concerning the leaky relu, the training process was certainly faster, though the results were less accurate so we decided to keep a sigmoidal activation for the network. We are pleased to say that our goals for the second defense were reached and even surpassed. We have a fully functional network that works for different processes of recognition. Still, something to enhance would be the recognition of hexa-decimal characters. We actually have an 80 percent rate of success and could certainly go up to 90 percent with a more accurate training dataset and a longer training process.

7 Parser, Solver and Display

7.1 Parser

First of all, we developed a feature allowing the user to give, as an input, a file containing an example of a sudoku with "." representing empty site. Therefore, we had to implement a parser.



```
⌥⌘1  armandblin@Armands-MBP:~/Desktop/ocr-sudoku/sudoc/solver
~/De/ocr-sudoku/s/solver  main ?1  cat sudok_1
... ..4 58.
... 721 ..3
4.3 ... ...

21. .67 ..4
.7. ... 2..
63. .49 ..1

3.6 ... ...
... 158 ..6
... ..6 95.
~/De/ocr-sudoku/s/solver  main ?1  ./solver sudok_1
000 004 580
000 721 003
403 000 000

210 067 004
070 000 200
630 049 001

306 000 000
000 158 006
000 006 950
```

Figure 32: Parser example

7.2 Sudoku validity check

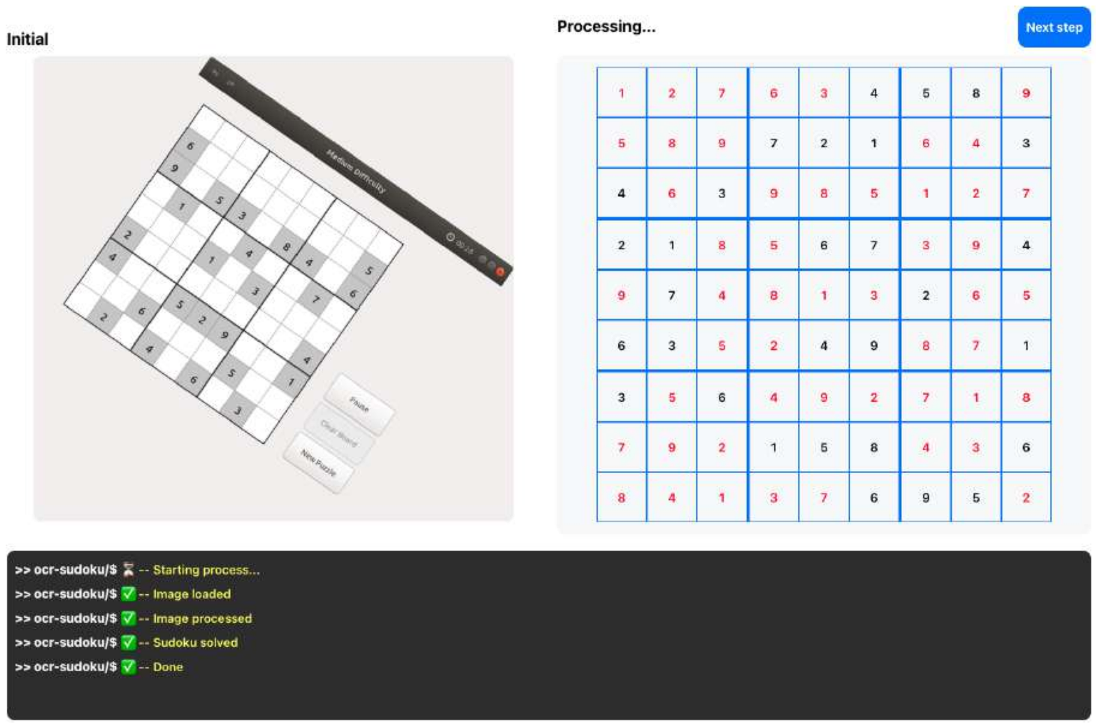
Before looking for the sudoku solution, we check if there is any inconsistency that would make the sudoku unsolvable like two identical numbers on the same row, column or in the 3x3 section. If the sudoku is impossible then it returns an error. Otherwise, when trying to solve it, if it goes through all the possibilities and finds no solution then it returns an error.

7.3 Solver

The solver has been implemented through the use of "backtracking", it is a method called "brute force" that will test all possibilities by recursion and thus find a solution.

7.4 Result display

For the moment the choice of the display has not been made, that's why we have created a simple display function allowing to display the solved sudoku with spaces to represent the grids. Here is an example:

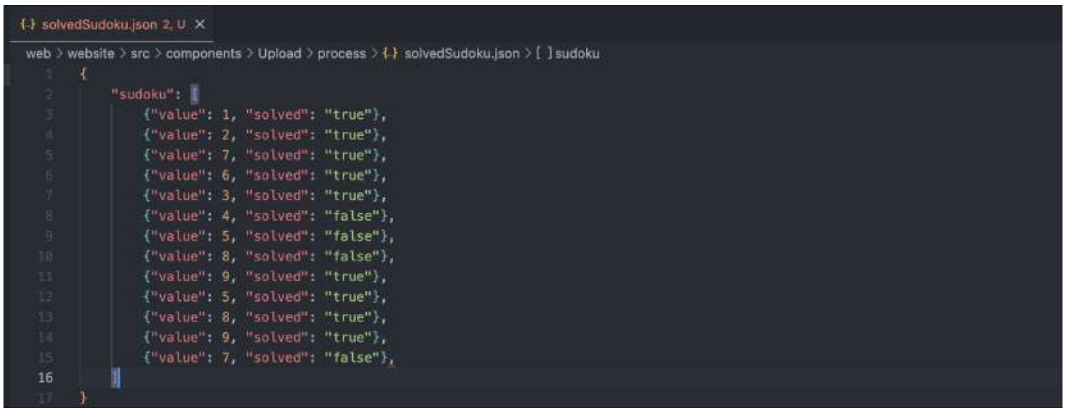


1	2	7	6	3	4	5	8	9
5	8	9	7	2	1	6	4	3
4	6	3	9	8	5	1	2	7
2	1	8	5	6	7	3	9	4
9	7	4	8	1	3	2	6	5
6	3	5	2	4	9	8	7	1
3	5	6	4	9	2	7	1	8
7	9	2	1	5	8	4	3	6
8	4	1	3	7	6	9	5	2

```
>> ocr-sudoku/$ ⏳ -- Starting process...
>> ocr-sudoku/$ ✅ -- Image loaded
>> ocr-sudoku/$ ✅ -- Image processed
>> ocr-sudoku/$ ✅ -- Sudoku solved
>> ocr-sudoku/$ ✅ -- Done
```

Figure 33: Solved sudoku grid

Furthermore, in order for the API to work we need to transmit a .JSON file which can give the solution of the sudoku. We also add the possibility to know if the number has been marked by the solver to have a better display. Here is an example:


{ } solvedSudoku.json 2, U
web > website > src > components > Upload > process > { } solvedSudoku.json > [ ] sudoku

```
{
    "sudoku": [
        {"value": 1, "solved": "true"},
        {"value": 2, "solved": "true"},
        {"value": 7, "solved": "true"},
        {"value": 6, "solved": "true"},
        {"value": 3, "solved": "true"},
        {"value": 4, "solved": "false"},
        {"value": 5, "solved": "false"},
        {"value": 8, "solved": "false"},
        {"value": 9, "solved": "true"},
        {"value": 5, "solved": "true"},
        {"value": 8, "solved": "true"},
        {"value": 9, "solved": "true"},
        {"value": 7, "solved": "false"},
    ]
}
```

Figure 34: .JSON File

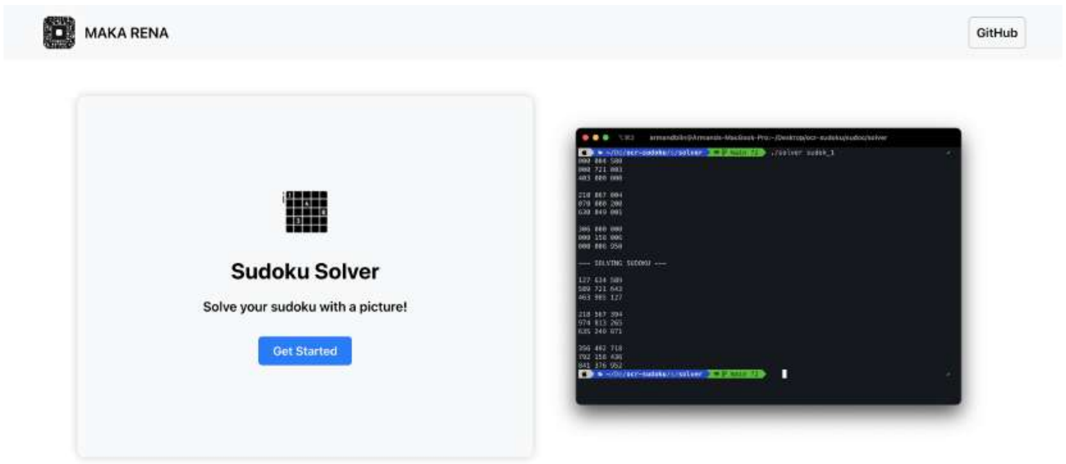
8 Website API

8.1 Webiste

We wanted to have an UI running on a Website instead of GTK. This website is a ReactJS website which allow use to use JavaScript to manage easily variables and the process of our OCR program.

8.1.1 Homepage

On the homepage you will be able to see a quick explanation of the project and a button to go to the process section.



```
./solver sudok_1
000 004 580
000 721 003
403 000 000

210 067 004
070 000 200
630 049 001

306 000 000
000 158 006
000 006 950

---- SOLVING SUDOKU ----

127 634 589
589 721 643
463 985 127

218 567 394
974 813 265
635 249 871

356 492 718
792 158 436
841 376 952
```

Figure 35: Homepage Section

8.1.2 Upload picture

On this section, you can upload the sudoku picture to solve.

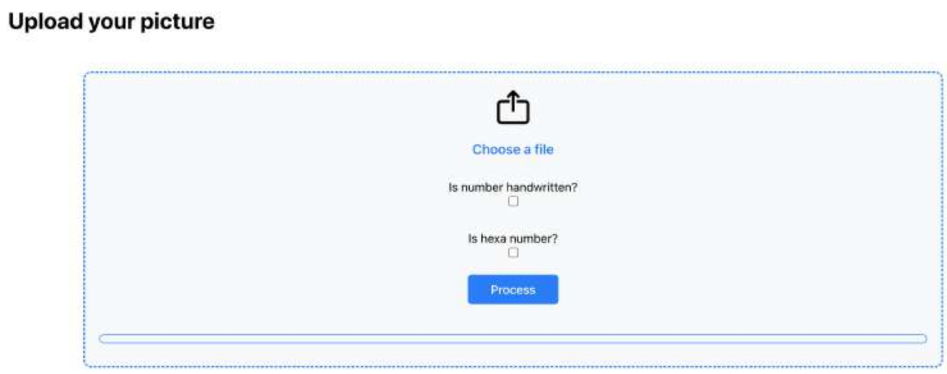
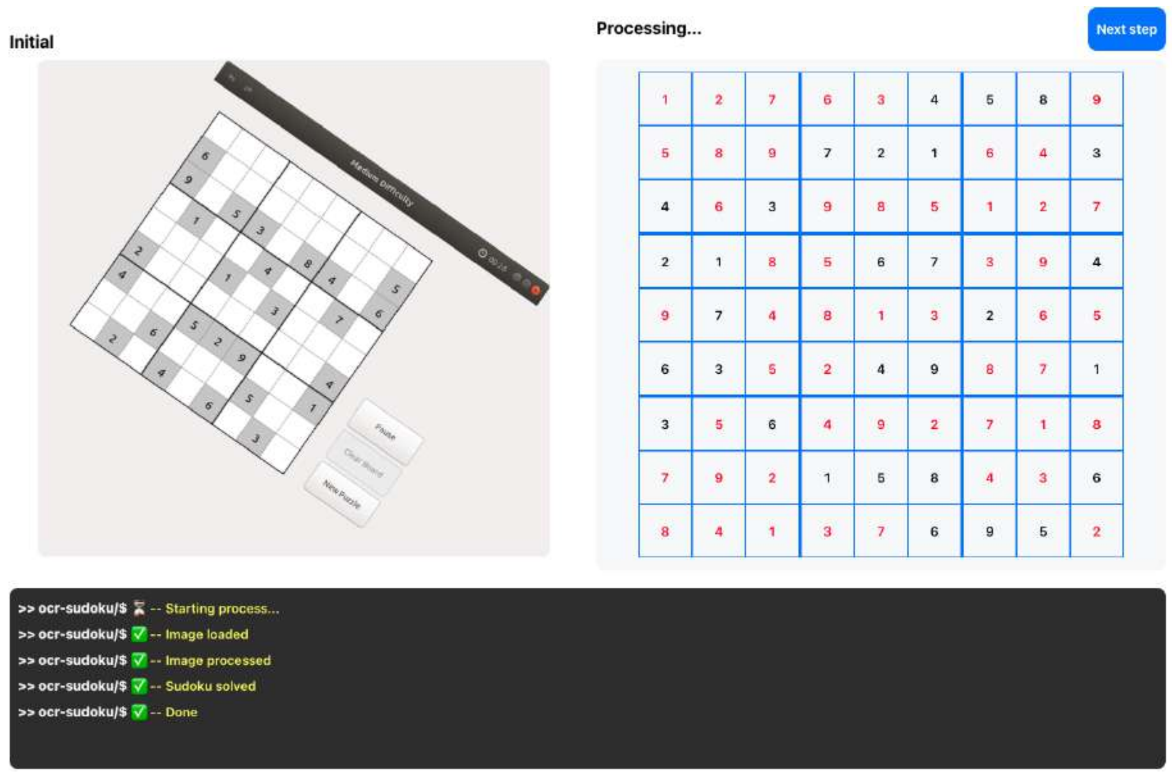


Figure 36: Upload Section

8.1.3 Result section

Here is the result section, where you can see the different steps of the process.



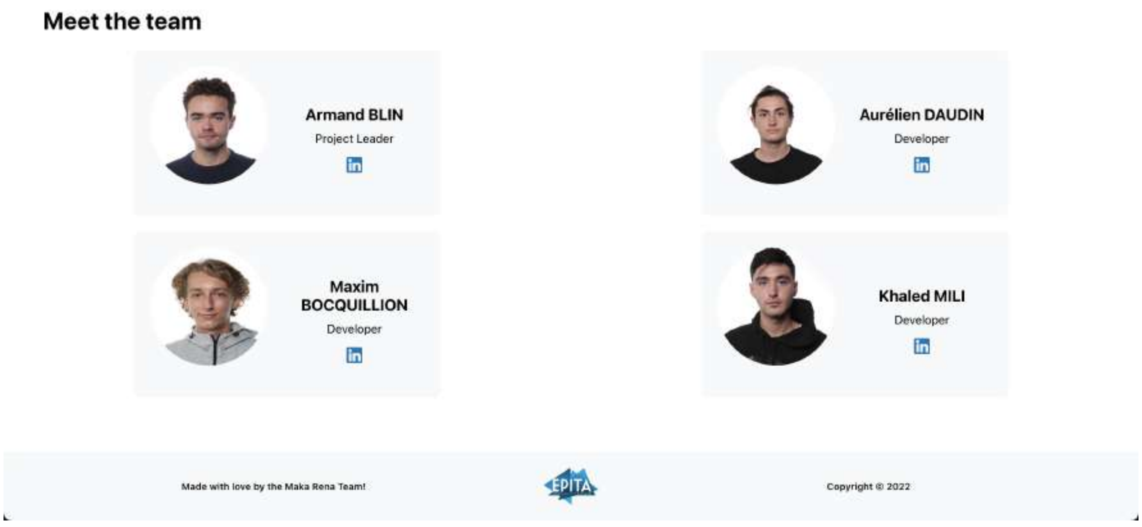
1	2	7	6	3	4	5	8	9
5	8	9	7	2	1	6	4	3
4	6	3	9	8	5	1	2	7
2	1	8	5	6	7	3	9	4
9	7	4	8	1	3	2	6	5
6	3	5	2	4	9	8	7	1
3	5	6	4	9	2	7	1	8
7	9	2	1	5	8	4	3	6
8	4	1	3	7	6	9	5	2

```
>> ocr-sudoku/$ ⏳ -- Starting process...
>> ocr-sudoku/$ ✅ -- Image loaded
>> ocr-sudoku/$ ✅ -- Image processed
>> ocr-sudoku/$ ✅ -- Sudoku solved
>> ocr-sudoku/$ ✅ -- Done
```

Figure 37: Result Section

8.1.4 Team section

Finally, here is the presentation of the Maka-Rena Team.



Armand BLIN
Project Leader

Aurélien DAUDIN
Developer

Maxim BOCQUILLION
Developer

Khaled MILI
Developer

Made with love by the Maka Rena Team!

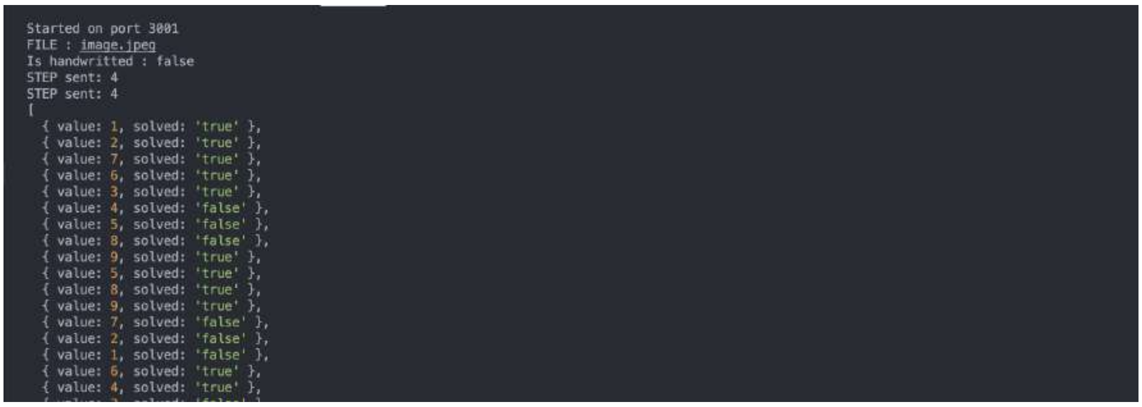
EPITA

Copyright © 2022

Figure 38: Team Section

8.2 API

We had to implement an API (on port 3001 by default) to begin and check the different steps of the process because only an API can execute command lines in a web architecture. We also had to implement the parsing of the result .JSON file. Here is an example of a few log of the API:



```
Started on port 3001
FILE : image.jpeg
Is handwritted : false
STEP sent: 4
STEP sent: 4
[
  { value: 1, solved: 'true' },
  { value: 2, solved: 'true' },
  { value: 7, solved: 'true' },
  { value: 6, solved: 'true' },
  { value: 3, solved: 'true' },
  { value: 4, solved: 'false' },
  { value: 5, solved: 'false' },
  { value: 8, solved: 'false' },
  { value: 9, solved: 'true' },
  { value: 5, solved: 'true' },
  { value: 8, solved: 'true' },
  { value: 9, solved: 'true' },
  { value: 7, solved: 'false' },
  { value: 2, solved: 'false' },
  { value: 1, solved: 'false' },
  { value: 6, solved: 'true' },
  { value: 4, solved: 'true' },
```

Figure 39: API Console Example

9 Global process of our OCR

Here is for the moment our global process :

- Treatment :
  - Grayscale
  - Gaussian Blur
  - Sobel Filter
- Grid Detection :
  - Hough Transform
  - Line detection
- Segmentation :
  - Splitting the picture into 81 cells containing number or blank
- Grid analysis :
  - Each picture is sent to the neural network
- Sudoku :
  - Is this grid solvable ?
  - Grid resolution
- End

10 Conclusion

Overall, we have implemented all aspects of this project. We are very happy with the result. We are able to take a picture of any Sudoku and solve it. We achieved our goal to have a fully functional OCR that can be used by anyone on any Sudoku.

List of Figures

1 Armand BLIN 3
2 Khaled MILI 4
3 Maxim BOCQUILLION 5
4 Aurélien DAUDIN 6
5 Before vs After Grayscale Filter 8
6 Before vs After Gaussian Filter 9
7 Before vs After Contrast Filter 9
8 Before vs After Black and White Filter 10
9 X Kernel for Sobel 10
10 Y Kernel for Sobel 10
11 Before vs After Sobel Filter 11
12 Before vs After Flood Filling 12
13 Before Manual Rotation 14
14 After Manual Rotation with angle $\theta = 35$ 14
15 Extract of the code of rotation 15
16 Image whith perturbations 15
17 Image representing in red all the lines in picture 1 16
18 Image representing in red all the lines in picture 2 16
19 Angle theta of the first line 17
20 Image representing in blue all the intersections in picture 1 18
21 Image representing in blue all the intersections in picture 2 19
22 Before vs After Resize 19
23 Before vs After Cleaning 20
24 All the cells numbered pictures 20
25 Xor Truth Table 21
26 Graphical Representation of Xor Neural Network 22
27 Implementation of the Forward Propagation 22
28 Implementation of the Backward Propagation 23
29 Result of Digit Conversion from Data-file to Matrix 24
30 Succes Rate Display of the Computer Character Prediction 25
31 Compression of data into CSV files 26
32 Parser example 28
33 Solved sudoku grid 29
34 .JSON File 29
35 Homepage Section 30
36 Upload Section 30
37 Result Section 31
38 Team Section 31
39 API Console Example 32

List of Tables

1 Task management tab 7