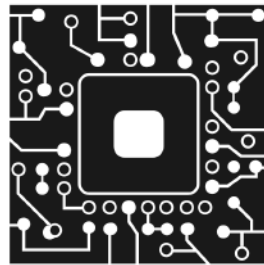


OCR - First Report

by MAKА-RENA

09 November 2022



Maka-Rena

Contents

1	The Team	2
1.1	Armand BLIN	2
1.2	Khaled MILI	3
1.3	Maxim BOCQUILLION	4
1.4	Aurélien DAUDIN	5
2	Task Management	6
3	Image Processing	7
3.1	Grayscale Filter	7
3.2	Gaussian Blur	8
3.3	Sobel	8
3.4	Problems encountered	9
3.5	Manual Rotation	10
4	Grid Detection	11
4.1	Line Detection (Hough transform)	11
5	Grid Splitting	12
5.1	Resizing	12
6	Neural Network	13
6.1	XOR Neural Network	13
6.1.1	Construction Steps	13
6.1.2	Implemented Libraries	13
6.1.3	Main Procedure	13
6.2	Character Recognition Neural Network	15
6.2.1	Neural Network Structure	15
6.2.2	Save and Load	15
6.2.3	Training Data	16
6.3	Next Steps	16
7	Parser, Solver and Display	17
7.1	Parser	17
7.2	Sudoku validity check	17
7.3	Solver	17
7.4	Result display	17
8	Global process of our OCR	19
9	Conclusion	19

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	First Defense
Image processing					
Image Loading			Principal	Second	100%
Color Removing			Principal	Second	90%
Filter the Image			Principal	Principal	70%
Rotation	Principal				90%
Grid detection			Second	Principal	40%
Image resizing			Second	Principal	50%
Digit recognition					
Neural Network		Principal			80%
AI Dataset		Principal			90%
Other					
Sudoku solving	Principal				100%
Graphic Interface	Principal				0%
Web site	Principal				0%

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 multiples 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

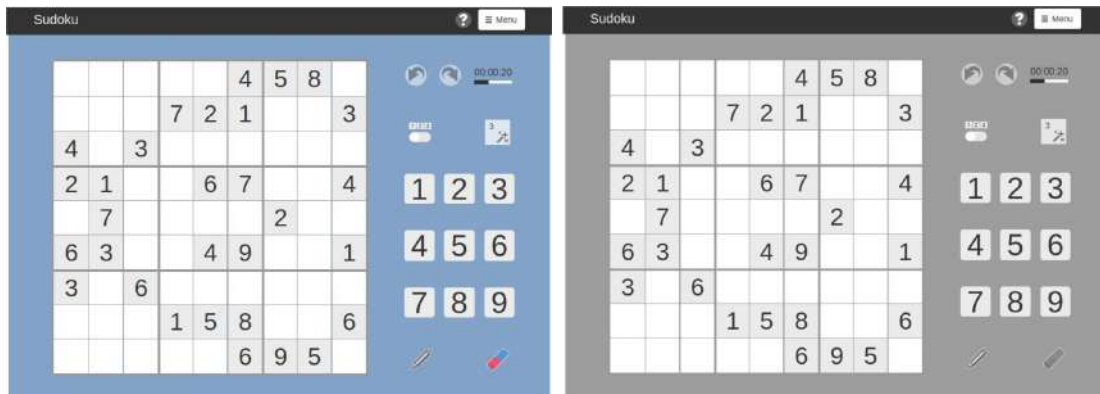


Figure 5: Before vs After Grayscale Filter

3.2 Gaussian Blur

Once that our picture is grayscale, 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.

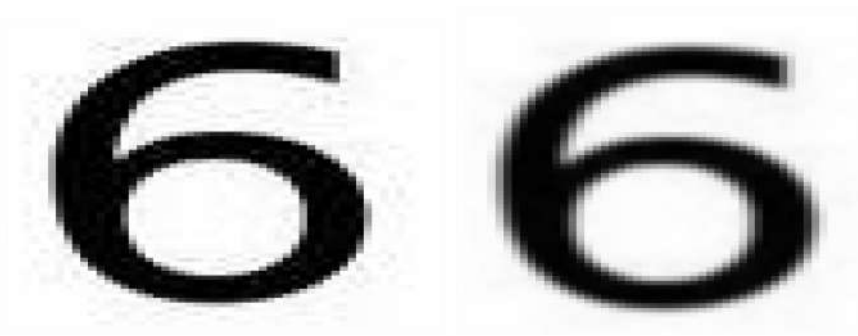


Figure 6: Before vs After Gaussian Filter

3.3 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 7: 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 8: Y Kernel for Sobel

These gradient values will tell us where are the brutal color variations. By re-scaling 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 $O = \text{atan2}(G_y, G_x)$, 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

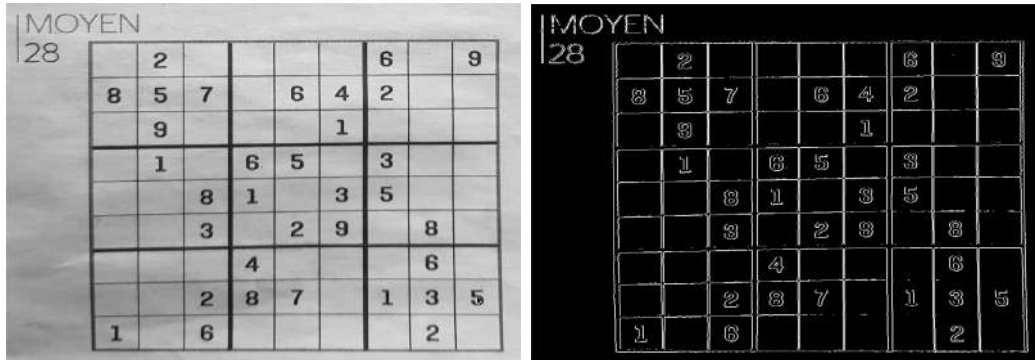


Figure 9: Before vs After Sobel Filter

3.4 Problems encountered

- 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. + explanation
- 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.

3.5 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.

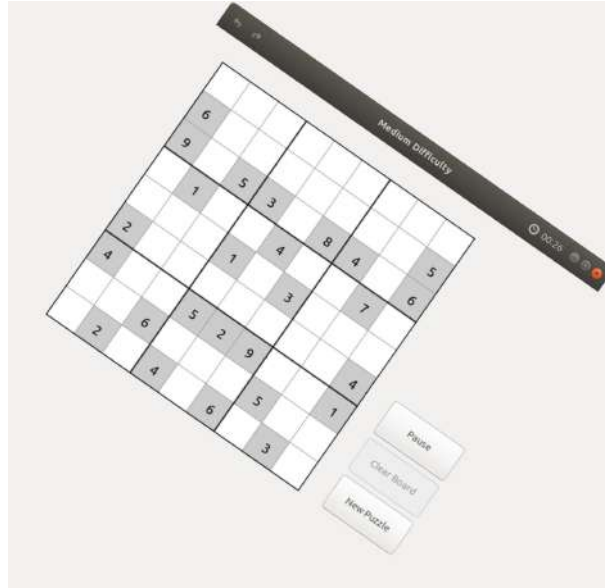


Figure 10: Before Manual Rotation

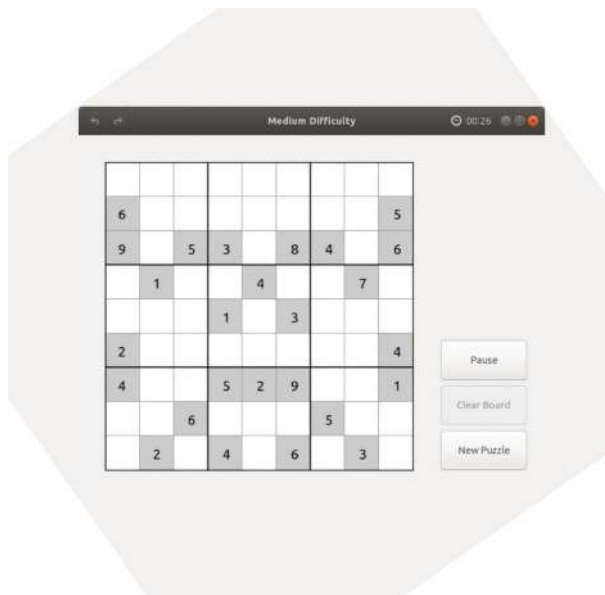


Figure 11: After Manual Rotation with angle $\theta = 35$

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 12: Extract of the code of rotation

4 Grid Detection

We need to analyse the image to detect the grid and to cut it into smaller images. These smaller images will be the cells that we will analyse with the Neural Network.

We are disappointed because we wanted the line detection part finished for this first defense. Due to three things we were no able to do it.

- Our first mistake was to split up the part about processing the image and the line detection but this part need the first one to be implemented.
- The second was that we had no idea where to start. We spent a lot of time on searching on internet to find the perfect algorithm and process to do it.
- The third one was the many errors on the filter part and all the optimizing part about choosing the best coefficients and numbers to a perfect result that took most of our time.

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.

5 Grid Splitting

When the grid will be detected, we will have to use these coordinated to process and solve this sudoku. To do so we will split the principal picture into 21 pictures each one representing a cell

5.1 Resizing

The Neural Network need a image containing only the digit in it. We need to resize the main picture to obtain a new one with only the cell containing the digit. The 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 will be called by the one finding the coordinates of the cells.

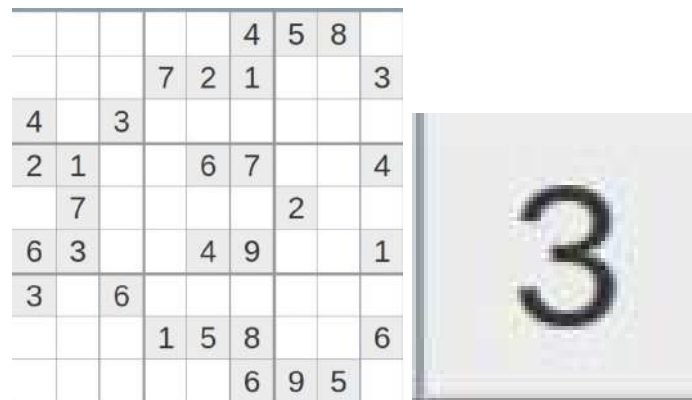


Figure 13: Before vs After Resize

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 14: 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.

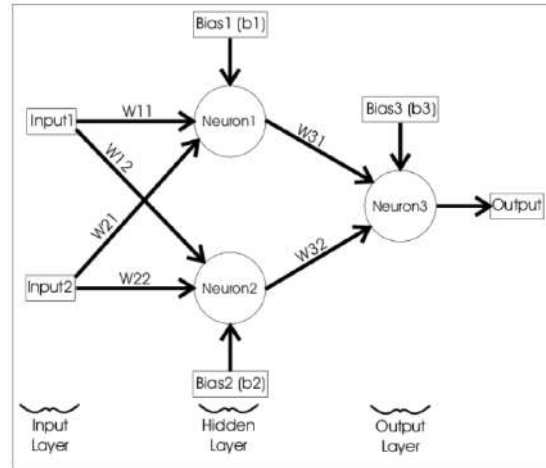


Figure 15: Graphical Representation of Xor Neural Network

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 16: 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 = subtract(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 17: 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.

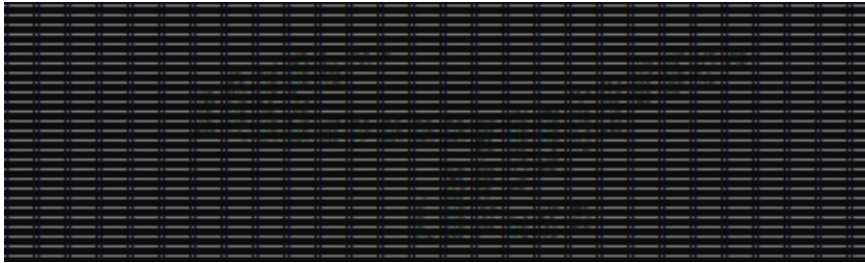


Figure 18: Result of Digit Conversion from Data-file to Matrix

6.3 Next Steps

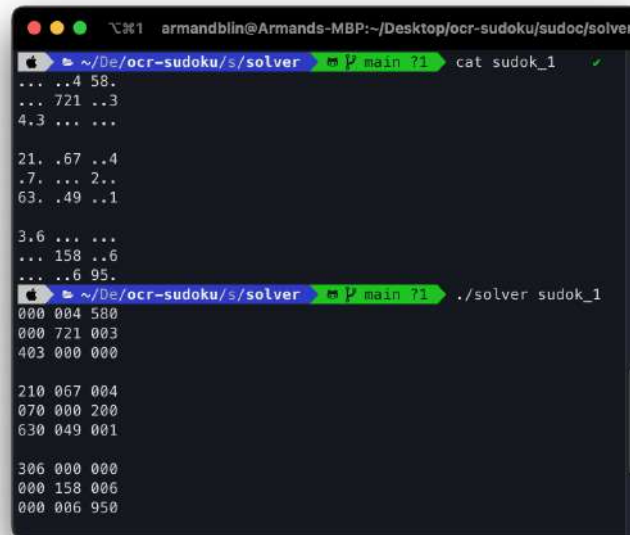
We are pleased to say that our goals for the first defense were reached and even surpassed. Still, even though our neural network can train well, we did not manage to implement the prediction process correctly for the moment. This is the main goal that we must reach for the last defense in December. Of course, we have other goals in mind to optimize our neural network. The principal ones are :

- Adding biases to improve and perfect the predictions of the network.
- Implement a Leaky Relu function to also adjust the predictions.
- Use a Softmax algorithm to get a better probability distribution for the output layer.

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.

A terminal window titled 'armandblln@Armands-MBP:~/Desktop/ocr-sudoku/sudoc/solver' shows the execution of a parser. The command 'cat sudok_1' is run, displaying a 9x9 grid of numbers and dots. Then, the command './solver sudok_1' is run, displaying the same grid with each cell's value converted to a 3-digit decimal representation (e.g., 580 for 58, 003 for 3, etc.).

```
armandblln@Armands-MBP:~/Desktop/ocr-sudoku/sudoc/solver$ cat sudok_1
... ..4 58.
... 721 ..3
4.3 ... ...

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

3.6 ... ...
... 158 ..6
... ..6 95.

armandblln@Armands-MBP:~/Desktop/ocr-sudoku/sudoc/solver$ ./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 19: 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.

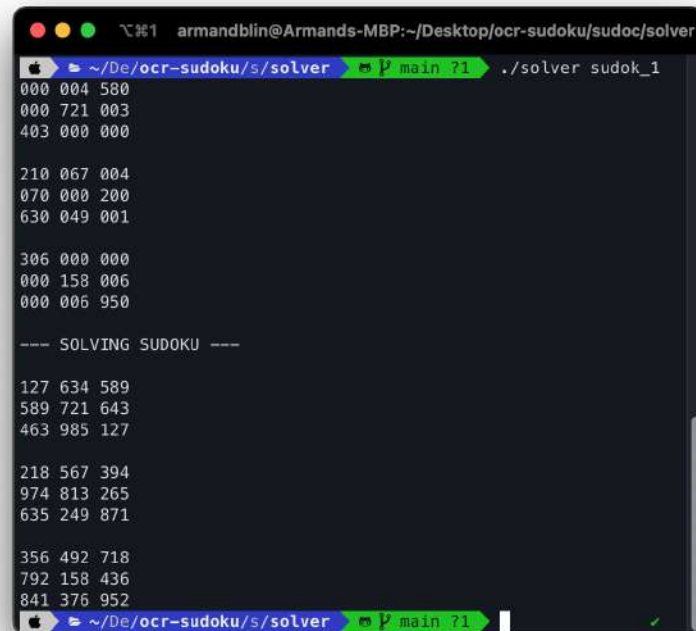
7.3 Solver

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:



```
armandblin@Armands-MBP:~/Desktop/ocr-sudoku/sudoc/solver
~/De/ocr-sudoku/s/solver main 71 ./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 20: Solved sudoku grid

8 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

9 Conclusion

Overall, we have already implemented several aspects of this project. We are very happy with this first result and we are looking forward to continue this project to have a successful program by December.

List of Figures

1	Armand BLIN	2
2	Khaled MILI	3
3	Maxim BOCQUILLION	4
4	Aurélien DAUDIN	5
5	Before vs After Grayscale Filter	7
6	Before vs After Gaussian Filter	8
7	X Kernel for Sobel	8
8	Y Kernel for Sobel	8
9	Before vs After Sobel Filter	9
10	Before Manual Rotation	10
11	After Manual Rotation with angle $\theta = 35$	10
12	Extract of the code of rotation	11
13	Before vs After Resize	12
14	Xor Truth Table	13
15	Graphical Representation of Xor Neural Network	14
16	Implementation of the Forward Propagation	14
17	Implementation of the Backward Propagation	15
18	Result of Digit Conversion from Data-file to Matrix	16
19	Parser example	17
20	Solved sudoku grid	18

List of Tables

1	Task management tab	6
---	-------------------------------	---