

ACE OCR PROJECT

Group Name: ACE

Project Name: OCR Sudoku Solver

Duration: September 2021 – December 2021

Group Members:



Erwin Rodrigues



Bianca Onciul



Carl Kone Takuefou



Mehdi Amrani

Table Of content

- 1.) Introduction**
- 2.) Task Distribution**
- 3.) Actual State of the project**
- 4.) Technical aspects of each part.**
 - 4.1) Image preprocessing**
 - 4.2) Neural Network**
 - 4.3) Solver Program**
- 5.) Conclusion**

1.) Introduction

In the framework work of our 3rd semester project, our groupe has to realise, in less than 4 months, a Optical character recognition program that is capable of solving a Sudoku grid given to it as input and produce an output image of the solved grid.

Underneath the above simple phrase lies a lot of intermediate steps to quit from the input to the output, a lot of cases of images that can be given as input to handle algorithmically and a lot of brainstorming in order to resolve any grid, handle the storage, recognise any form of the digits, clean the the image given, and the list goes on 🙄.

It soon became necessary to efficiently split the task in order to handle all the sub-problems to our main problem. Below is a representation of the Task distribution, the assigned member(s) for each task, and a percentage of completion of each task.

2.) Task Distribution for the first presentation

Tasks	Sub Task 1	Sub Task 2	Sub Task 3	Sub Task 4	Assigned Member	TOTAL Percentage complete
Sudoku Solver Algorithm	Solver Implementation	Read grid from file	Write solved grid in file	Detect unsolvable grids	Carl kone Takuefou	100 %
% complete of subtasks	100 %	100 %	100 %	100 %	100 %	
Image Preprocessing	Image load and colour suppression	Manual rotation of Image	Grid detection	Separation of each cells into individual images	Bianca Onciul Carl kone Takuefou	80 %
% complete of subtasks	100 %	100 %	70 %	70 %	//	
Neural network	Proof of concept with XOR	Store weight in a file	//	//	Erwin Rodrigues Mehdi Amrani	100 %
% complete of subtasks	100 %	100 %	//	//	//	

3.) Actual State of the project.

As can be seen above, Our groupe has greatly advanced on the project. We have even done some of the tasks due for the second presentation. Below are all the technical details needed to understand Each tasks.

4.) Technical aspects of each part.

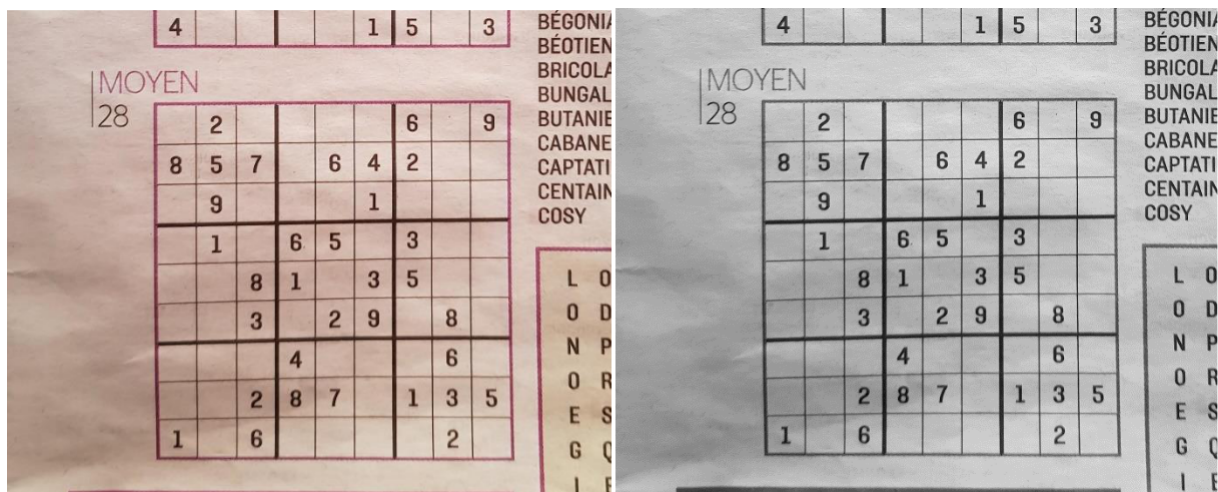
4.1) Image preprocessing

Image (ONCIUL) :

For the first defence of the OCR project, I was in charge of the pre-processing of the image. The pre-processing of the image is needed in order to be able to detect the grid cell positions and split the sudoku image.

For the image, I needed to use the SDL library, and I used the functions to get the pixels, the rgb values and set the pixels from the 3rd programming tp.

For the pre-processing, I had to start with the grayscale filter, so I used the one that I had done for the 3rd programming tp.



Original image

Grayscale image

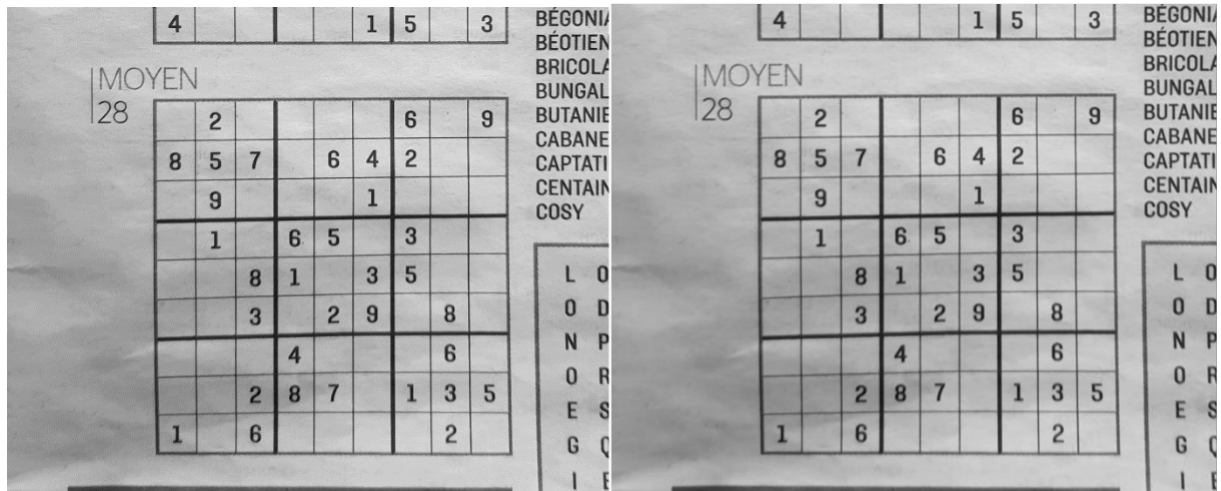
Then I really started the pre-processing by creating a program to apply a Gaussian blur.

The Gaussian blur is a filter that when applied on a grayscale image removes the noise by removing isolated pixels. In order to create this program, I read multiple articles and ended up finding a clear video by computerfile explaining how the Gaussian blur works. First, I needed a kernel, which is a matrix with values that will be multiplied with pixel values from the image. I found an example of a kernel on Wikipedia and created a global array with its values.

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

The kernel for the Gaussian blur

Then I needed a global variable with the sum of the integers of the array. I had to go through all the pixels of the image, and for each pixel (x,y) I created an array storing the grayscale value of the surrounding pixels. The coordinates of the surrounding pixels go from y-2 to y+2 and x-2 to x+2, in order to create an array of the same length as the kernel. I multiplied each pixel from the array with the corresponding value of the kernel, then I added up all these values and divided the result by the sum of the integers of the kernel. I returned this result which is the new grayscale value of the pixel at coordinates (x, y). I put this new value on another surface so that I wouldn't modify the values on the original picture, which would have distorted the calculations for the Gaussian blur.



I had many errors in my code in C, such as a segmentation fault. Since I couldn't find how to debug the program, I wrote it in C# and was able to debug it on Riders. I still had problems with the types and the SDL elements in C, therefore it took me a lot of time to solve the problems.

After that, I started the contours detection, by using Sobel filter. Sobel filter works on the same principle as the Gaussian blur. After reading several pages, I also found a clear video by computerfile explaining the principle of the Sobel filter. This time, two kernels are needed, one to find the gradient on the x abscissa, and another one to find the gradient on the y abscissa.

$$\mathbf{G}_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

Kernel for gradient on x abscissa

$$\mathbf{G}_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

Kernel for gradient on y abscissa

Then I had to go through all the pixels of the image again, and for each pixel (x,y) I had to store the grayscale values of the surrounding pixels in an array again. The coordinates of the surrounding pixels go from y-1 to y+1 and from x-1 to x+1, in order to create an array of the same length as the kernels. I multiplied each pixel from the array first with the corresponding values of the x kernel and added up these values, then with the values of the y kernel, and added up these values too. I then found the resulting gradient by computing its norm. The gradient found is the new grayscale value of the pixel (x,y). I then put this pixel on another surface so that it wouldn't modify the values on the original picture, which would have distorted the calculations for the Sobel filter. The input image of the Sobel filter is the image on which the Gaussian filter have been applied.

This time, I wrote the program in C# first, in order to avoid errors such as an index out of range and to be able to debug it, then I wrote it in C. However, the grayscale values of the pixels of the image are false in the C program. They are way too often equal to 255 even when the color is supposed to be darker. I unfortunately have not been able to solve this problem yet, even after long hours spent searching for the solution.

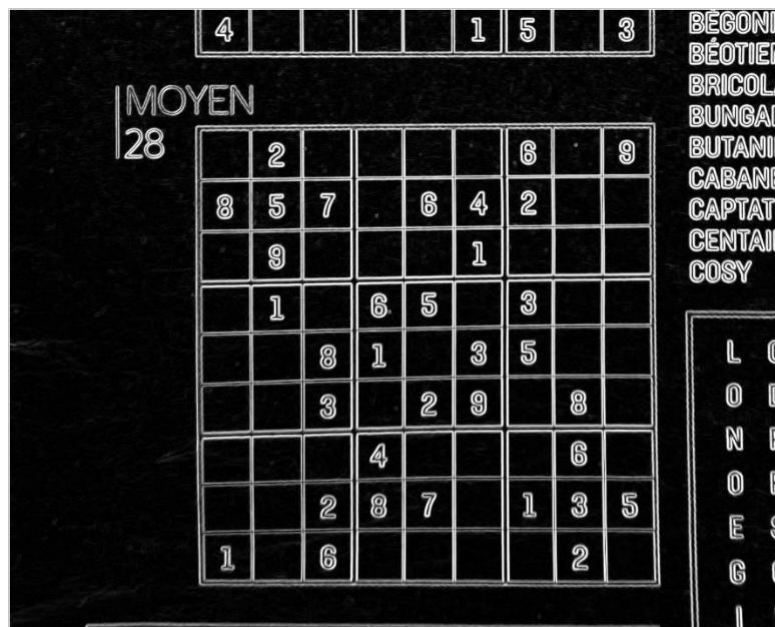
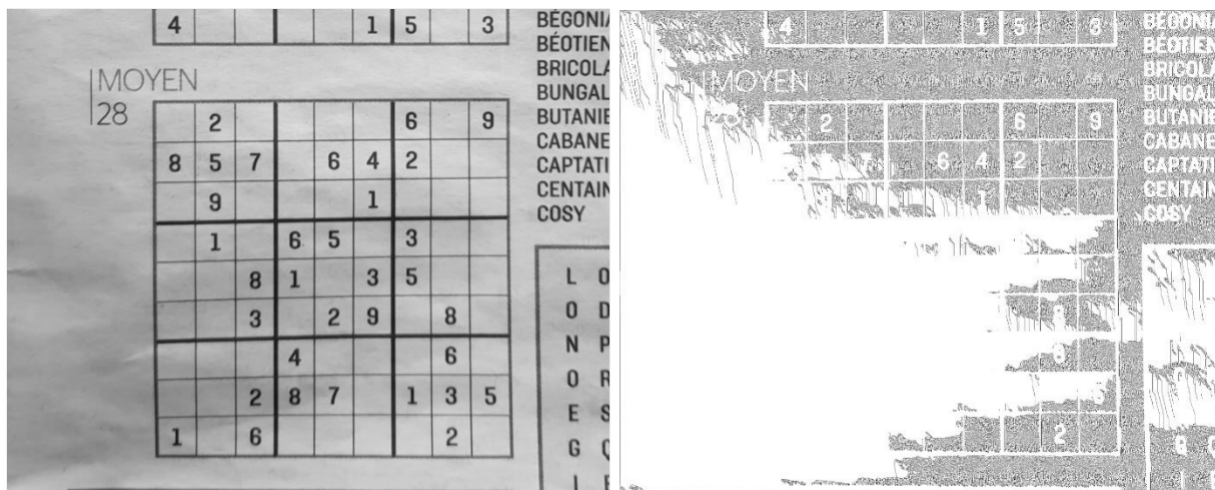


Image expected in C (obtained in C#)

While trying to solve the problem for the Sobel filter, I started my research for line detection using Hough transform.

Hough transform is a program that takes an image on which was applied a Canny or Sobel filter, and returns a matrix with the polar coordinates for each possible line in the image. This matrix allows to find the lines in the image. I have started writing the program. I created a matrix to store the possible coordinates for each pixel of image. Then, I had to iterate through all the pixels of the image. For each pixel, I checked if it was of color white, and if it's the case, then for each theta value going from 0° to 180° , I found the rho value for that specific point. With rho and theta, I found the coordinates in the matrix and incremented by one. After executing the program, the highest values in the matrix represent the polar coordinates of the lines in the image.

4.2) Neural Network

Neural Network (RODRIGUES):

For the first defense, of our S3 Project OCR, I was tasked to work on developing the proof of concept for our neural network, which needed to be able to learn the XOR logic gate.

On its own, the way a neural network behaves isn't that hard to comprehend. It basically has an input layer, hidden layers, and an output layer all linked with weights and biases. Thanks to the link provided in the Project Specifications, finding material to learn more about it wasn't tedious.

Nevertheless, working with the provided python code was not efficient at all, python being a higher-level language compared to C. Using one language and turning to the other, without extensive use of libraries, wasn't quite possible.

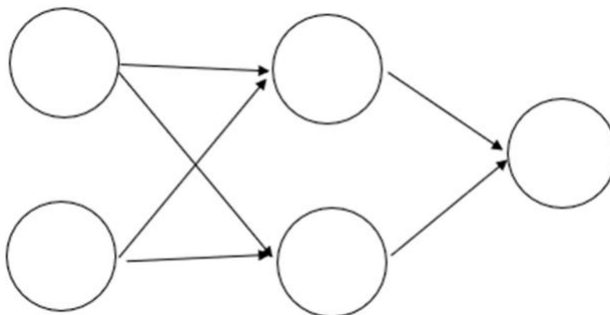
And so, after watching a couple of videos talking about the implementation of a neural network, I started to browse through a dozen of forums and documentation in order to find what I could and should use and what should be avoided, like the TensorFlow which is prohibited.

I had quite some trouble adapting for C# to C, without all the types and possibility for creating classes in C, I took a fair amount of time.

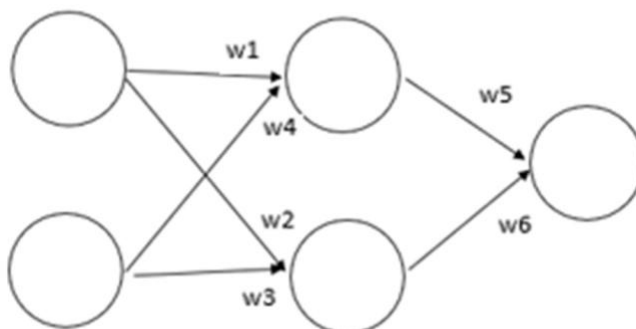
Basic presentation of the neural network's implementation:

For this proof of concept :

- The number of layers and neurons for each layer are predefined,
We have 3 layers, namely the input layer, the hidden layer, and the output layer
The first 2 have 2 neurons each whereas the last one has only 1 neuron as can be seen below:



- We this network defined; the program creates outgoing weights from one neuron to every other neuron on the following layer:



- The inputs that we want the network to train for are also predefined by combinations of 0 and 1: 00 - 01 - 10 - 11
- Before the training begins, we specify the desired output, in our case for the XOR logic gates the outputs would respectively be: 1 - 0 - 0 - 1
- The training then starts, using back propagation, activation formulas and forward propagation the network adapts each weight accordingly to get a higher percentage of success and learn the correct/required output.

After some trial and error, the network finally works.

This being done, I then moved forward and started to implement a saving and a loading feature for the final weights of each neuron.

The saving feature is working just fine, iterating through the weights after the training is done and after being tested by the user, each weight is saved in a separate file called “saved_weights” that can easily be accessed.

On the other side, the loading feature is still a work in progress but is very close to be done with as well.

Screenshots of the neural network:

Below, you can see that the neural network is created.

```

---Creating Neural Network---

- Layers created

- Assigning weights

-> Weight : 0.840188
-> Weight : 0.394383
-> Weight : 0.783099
-> Weight : 0.798440
-> Weight : 0.911647
-> Weight : 0.335223

-----

Neural Network Created

```

The program prompts the user to enter the desired output for each set of input, it is able to learn each logic gate.

```
Enter the output corresponding to the size_tput: 0 0
1
Enter the output corresponding to the size_tput: 0 1
0
Enter the output corresponding to the size_tput: 1 0
0
Enter the output corresponding to the size_tput: 1 1
```

Next, we can see when the network is training and when the training is done:

```
Updated weight => 2.402864
Updated weight => 1.626323
Updated weight => 4.005155
Updated weight => -2.075291
Updated weight => 2.405201
Updated weight => 1.621702
Updated weight => 2.402864
Updated weight => 1.626323
Updated weight => 4.005155
```

```
Training Done!
```

Here, we can see the tests made by the user (the user has to type in the inputs to test with)

```
Input:
0 0
Output: 1

Input:
1 0
Output: 0

Input:
0 1
Output: 0

Input:
1 1
Output: 1
```

Finally, the weights are saved and can be accessed as seen below.

```
Input:
0 0
Output: 1

Weights successfully saved
rodrigues@DESKTOP-GPDM3J9:~/Epita/neural_net$ cat saved_weights
3.141526
2.090822
3.141538
2.090885
5.318061
-2.760080
```

For the next defense, my main goal will be to implement the neural network that will be able to recognize the numbers fed to it, and possibly try to do a website to showcase our project.

4.3) Solver Program

Solver Program (CARL KONE)

For this part of the project, which could be seen as the heart of the whole program, it was required to implement the algorithm that would take in a text file as input, resolve the grid contained in the text file and save the result in another text file having a « .result » extension.

For the first part, it was necessary to be able to read the file. And also to Make sure that the file contains text in the valid grid format.

The grid below:

					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	

Can be represented in a text file as shown below:

```
... ..4 58.
... 721 ..3
4.3 ... ...

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

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

After Writing the function that can read the text file displayed above. Now came the most difficult part, Solving the grid.

I used a 2-D array in other to represent the string as integers to ease manipulations. (‘.’ are represented by 0).

I Wrote a lot of intermediate functions in order to verify that a row, a column and a 3by3 square is solved or not according to Sudoku rules.

Then Came the core recursive function that used back tracking in order to fill the 2-D array with correct integer values.

Below is a a text gird that is not yet solved:

```
53. .7. ...  
6.. 195 ...  
.98 ... .6.  
  
8.. .6. ..3  
4.. 8.3 ..1  
7.. .2. ..6  
  
.6. ... 28.  
... 419 ..5  
... .8. .79
```

And below the solution of the grid after running the program:

SOLVED GRID BELOW:-----

```
534 678 912  
672 195 348  
198 342 567
```

```
859 761 423  
426 853 791  
713 924 856
```

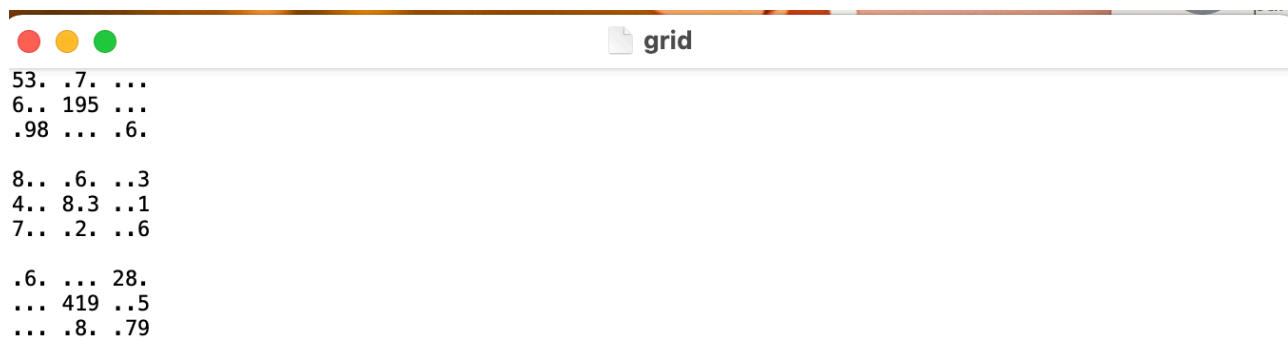
```
961 537 284  
287 419 635  
345 286 179
```

Storing the grid in a file.....

Program ended with exit code: 0

After that. It Was necessary to store the result of the grid in a « .result » text format. So I wrote a function that does Exctly that as you can see below.

The « grid » file:



The screenshot shows a file editor window with the title 'grid'. The window contains the following text:

```
53. .7. ...  
6.. 195 ...  
.98 ... .6.  
  
8.. .6. ..3  
4.. 8.3 ..1  
7.. .2. ..6  
  
.6. ... 28.  
... 419 ..5  
... .8. .79
```

The « grid.result » file:



5.) Conclusion

Overall, the group has made a great advance on the project. The solver and proof of concept of the neural network's implementation are done. The preprocessing of the image is also almost done too terminated.

Some work has also been made towards the 2nd defense already and we are on track to finish even all the bonuses.