

Final Report

OCR

Optimal Character Recognition

HELME-GUIZON Claire

MORIN Guillaume

STARCK Sophie

ZAMATTIO Emanuele

Promo 2020

7 December, 2016

1 Acknowledgments

We would like to thank all the members of our group, Sophie, Guillaume, Emanuele and Claire, for having worked so hard on this project, and giving up so much of their free time.

We would also wish to express our gratefulness to both Pierre-Louis and Nathan, our assistants. They have taught us, with a lot of patience, how to code in C, our chosen programming language for the project. Marwan Burelle's algorithm classes were also a big help. They helped us learn how to structure our code and debug it.

Other than our assistants and our teachers, our friends have also been a great help in times of trouble. Many times we felt helpless and misunderstood our issues. We would like to thank them all for being there. A closer attention goes to Bastien Griffet who was present whenever we had problems with Git-hub, or when we were stuck on a piece of non-compiling code.

Finally, we would like to thank Epita for giving us the opportunity to work on such a project, which made us realize what team-work is all about.

Summary

1	Acknowledgments	1
2	Introduction	3
3	What is an OCR program ?	4
4	The Members of the Team	5
5	Task distribution	9
6	Image Processing	13
7	Block, lines and characters detection	16
8	Neural Network	22
9	Graphical User Interface	27
10	Difficulties Encountered	29
11	Definitions	31
12	Conclusion	32

2 Introduction

This report's purpose is to present our OCR project and how it evolved from its beginning to the end. During the first part of the year, every second year student of EPITA had to team up to start this project. Our group was formed quite easily, we met at the beginning of the year and got along well so we decided we would be a good work team. That is how the GCS team was created.

We all had one project experience before, we created a video game throughout our first year as EPITA students. Even though it taught us a lot, regarding rigor, organization and group management, this project is completely different on many levels.

First of all, there is no big software used to help us implement it, meaning that everything created will only be coded by our little hands and brains. Which, in our opinion, considerably improved our knowledge in C.

Secondly, an OCR is a complex system that we did not quite understand until now. So a huge amount of research had to be done beforehand. Moreover, it is segmented in different type of jobs.

Having finished this project, we realize we have learned a lot. We noticed that communication is essential if we want the work to be done successfully, indeed, since every task is distributed between all the members of the group, we absolutely cannot work alone without noticing the others. Moreover, even if it is not perfect we are very proud to present the outcome of our very first software.

3 What is an OCR program ?

OCR stands for Optical Character Recognition. It designates the recognition of printed or written text in images, by a computer. To do so, the computer will have to match character images to an electronic version that corresponds to it.

The process can be decomposed in different steps :

- Reading the text, character by character
- Analysis of the scanned image
- Translation of the character image into character code

However, being able to scan different languages and hand-written text is really difficult, as the amount of data to recognize a single letter is quite significant. To handle this, we generally build OCR using neural networks.

OCR is heavily used in scanning domains (direct conversion from image to text document) and text edition.

4 The Members of the Team

4.1 Guillaume Morin

My name is Guillaume Morin, I am 18 years old and I was born in Le Kremlin-Bicetre. I'm in second year at EPITA, in International Section.

I have always been interested in programming, and new technologies in general. As my father is web-developer, I have been into this since I was young. I really like the fact that you can just do anything you want, if you put some efforts in it. Except that, I usually spend my free time listening to music (i like almost everything, I'm not that complicated), and I'm also fan of video-games.

I'm totally new to C programming, so it has been quite hard to begin. I had some experience before joining EPITA, but it was more about Python and Javascript. However, I think once you have spent a couple of hours manipulating functions on a same file, you begin to be comfortable with some notions, and it just goes better and better. Even if pictures manipulation isn't the kind of things I prefer, I really think this project is interesting, and I'm having a very good time working on it.

4.2 Sophie Starck

My name is Sophie Starck, I am 19 years old and I was born in Paris. I love to read and learn foreign languages, making my academic profile somewhat literary but I always had a particular taste for science, that is the reason why I pursued scientific studies. I am a student at EPITA, an engineering school specialized in computer science in the international section. I only started programming during my first year of EPITA but I have really become fond of it, I love the perspective it opens and the creativity it requires with an once of mathematical rigor.

This project is only my second experience in a team work, my first experience was very enriching and tough me a lot, but I think we always have more to learn, especially with different team-mates and a different project. I believe that positive comes out of every experience, that is why I was thrilled to work on this project.

I believe one of my best qualities is my enthusiasm. Even in stress and conflicts I always show a great smile, and carry my team in a good mood. Moreover, I can demonstrate some great capacities to put myself to work when the urge makes it feel.

4.3 Claire Helme-Guizon

My name is Claire and I was born in Paris on the 16th May 1997. I am currently a second year student at Epita, in the international section. The reason I was able to enter that section was that I lived for three years in England as a toddler. Knowing how to speak English has really helped me in my life, and is very important for an engineer.

The idea of going to an engineering school specialized in computer skills came to me quite late last year. Before that I wanted to go to a more general school, but later realized that computer skills really interested me. However, having chosen this branch quite late, my knowledge in programming was close to zero when I came to Epita. Thanks to the teachers and my classmates, I have however learned a lot since the beginning of the year and am very happy about my choice.

I believe that enthusiasm can bring as much as skills, I have to... Currently in second year at Epita, an IT engineering school. I feel like my first year has brought me knowledge and interest in computer science. Therefore, this project will greatly profit from it, as I have get used to all the different processes to aboard one correctly. Programming is no longer black magics, it makes sense more and more. However, working in team is very important for me. I feel like you learn even more from others.

Starting this huge and first project is quite scary since I'm still discovering programming. Yet, the perspective of learning so much in this domain and from my co-mates is reassuring and kind of exciting!

4.4 Emanuele Zamattio

My name is Emanuele Zamattio. I was born in Rome (Italy), where I'm attending my studies at La Sapienza University of Rome in the faculty of Computer Science. I'm here at Epita as an exchange student for one year. There's a difference between my faculty of Computer science and the studies at EPITA. In fact we study to become computer scientist, not engineers; we put more attention on teorical math and computer's technologies (we have less preparation in Physics and electronics for example). In addition to this, we also have a different university system based on a 3+2 years timeline.

The first contacts with the computer science's world was with videogames. During the high school I had the chance to learn programmation and this gave me the possibility to fullfill two of my desires : create and solve problems. Nothing motivate me as trying to make thing works, and programmation concern also the process of creating and inventing. In the last year of high school I decided to study computer science, a simple choice for me.

This was the first experience and project with the c language for me. I had experience with java, c++ and python, in particular with java and the object-oriented programming, so in the first I was a little bit uncomfortable in working without classes and objects. Also caused by some miss understanding with the exchange student's admistration, I was not able to enter the project in the start.

5 Task distribution

5.1 For First Presentation

We think proper documentation is very crucial in a project. We felt like it was no need to rush into a matter we did not handle well. Thus, we took our time to do research before starting this complex project, to that extent we preferred to present our work chronologically.

September & October : Research

Those first month were dedicated mainly to research. We discovered the project in September and none of us were familiar with the term OCR so we first had to study what type of program it was, what did it do and how it work.

What came out of this surface research was an even bigger question : the Neural Network. We had all heard of it, relatively by far, but the details were hazy. So we dug up into the subject and it frankly took quite some time. We read the documentation on several websites but mostly on :

<http://neuralnetworksanddeeplearning.com>

which was really helpful. It helped us identify the project's problematics and the several parts we had to deal with in this neural network which are :

- Sigmoid (sigmoid func, biais weight).
- Gradient
- Learning rate.
- Backpropagation algorithm.

September to November : Implementation

When this whole process of research was over, we could start implementing. But since our ideas were clear concerning the work that had to be done we did not lose time determining the tasks and distributing them. Furthermore, the members of group agreed with one another really fast and naturally.

Sophie took care of the image processing, Guillaume of the segmentation and Claire started the Neural Network by implementing the Xor Operator.

For Emmanuele, we weren't too sure of if he was suppose to help us or not. But now we will for sure find a piece of work for him next time !

The different steps within each task were set as below :

- **Image processing :**
 - Initializing SDL library
 - Binarize
- **Segmentation**
 - Block Segmentation
 - Lines detection
 - Characters detection
- **XOR Operator :**
 - Neural net for learning
 - Struct
 - Sigmoid
 - layers
 - Gaussian distribution for random

5.2 For Second Presentation

The main goal of this period was to transform our work in a program close to a software. The hardest part was to link all of our work together. Indeed our previous work was called separately.

November to December : Implementation perfection and finishing

We kept the same task distribution. We felt it was easier than starting over on a new code - even if we would have learned a lot. Emanuele joined the group, and took care with Sophie of the interface.

The different steps within each task were set as below :

- **Image processing :**
 - Rotation
- **Segmentation**
 - Block Segmentation
 - Lines detection
 - Characters detection
- **GTK**
 - Browse
 - Run code
 - Display text
- **Neural Network :**
 - File I/O stream
 - Save
 - Load
 - Image to input vector
 - Create target
 - Training

5.3 Progression Overview

Sophie	Image processing	90%
Guillaume	Text detection	100 %
Claire	Xor	100%
Teamwork	Neural net	98%
Emanuele	Graphical interface	100%

FIGURE 1 – Recap chart

Image processing : Even though we tried to implement rotation, by lack of time we choose to focus on part of the project that seemed more important to us. Anyway, the research were done, and Sophie had a code really close to work.

Text detection : As you have seen during the defense, the text detection is rather well segmenting.

Neural Network : We hoped some better result regarding the recognition. Yet, our network is behaving properly for training. With more time it could have been even better - isn't it always the case. But for our level it seems like not so bad.

Interface : It is doing the minimum ask, and we are aware of this. Nevertheless, it's working, and is doing what is asked.

6 Image Processing

For an OCR to have an optimal level of efficiency, the picture treated has to go through some changes before it is submitted to the program. Indeed, some pictures have many imperfections that can generate bugs in the recognition. The image processing part is, therefore, useful to erase these imperfections before treating the image. The best way to do so is to put the picture into black and white.

6.1 The SDL library

For convenience matter, we chose the SDL library for image processing. Indeed, many images format are supported.

6.2 GreyScale

The first step to this work is to put the image in Greyscale. To do so, every pixel of the image must be read and changed into it's luminance value. The luminance is the intensity of light emitted by the pixel. Each pixel is divided in red, green and blue components and each one of these colors has a different intensity according to the human eye. The relative luminance of the RGB components can be calculated with this formula :

$$Y = 0.2126R + 0.7152G + 0.0722B .$$

Every pixel is replaced by it's value and the image is thus in greyscale.

6.3 Binarization

We might think that the work is done but black & white is not the same thing as greyscale. We literally want our image to be in black and in white, this process is called binarization. In this process we read the image in the same way, pixel by pixel. For each pixel we have to determine whether we put it in black or in white, to do so we used a threshold set to 128. The RGB value for white is 255 and for black is 0. The threshold chosen was simply the mid value.

Dilution, coloration, filtration

Dilution, coloration, filtration

La dilution

Cette étape consiste à ramener le degré naturel d'un whisky, souvent parfois plus de 60% vol., à un degré minimum de 40% vol., la force minimale en dessous de laquelle un alcool de grain ne peut être légalement appelé whisky. La dilution concerne la majorité des whiskies disponibles sur le marché. Elle s'effectue par adjonction progressive d'eau déminéralisée qui provoque la précipitation et la concentration des acides gras présents dans l'alcool. En plus de cette réaction chimique, la dilution modifie les arômes et les saveurs du whisky.

La coloration

Dans les années 1960, dans un contexte de forte croissance, les blenders ont peu à peu remplacé le verre transparent des bouteilles de whisky par du verre teinté. L'impact visuel de leurs facettes, en particulier la couleur et la limpidité

Longtemps ignorées des consommateurs, les pratiques de dilution, de coloration et de filtration sont devenues des sujets particulièrement sensibles. Depuis quelques années, le débat entre, d'une part, les consommateurs et, d'autre part, les distillateurs et les embouteilleurs indépendants, amène à se justifier sur l'intérêt de telles pratiques.

canon, de type 1750, employé pour la coloration de boissons fortement alcoolisées. Le caramel est introduit après la dilution du whisky, juste avant sa filtration.

Si l'assemblage de dizaines de fûts entre eux (blends) permet d'obtenir une constante en termes d'arômes et de goûts, la couleur résultant de l'assemblage n'est pas nécessairement la même d'un batch à un autre. Le changement de teinte d'un whisky, si léger soit-il, tend à provoquer une « crise de confiance » chez les consommateurs peu avertis. Pour beaucoup, la variation de couleur implique nécessairement un changement de qualité, ce qui est loin d'être toujours le cas. Un whisky de couleur ambrée et couleur flûte plus facilement l'œil du consommateur qu'un whisky arborant une teinte légèrement dorée. Même si « la couleur ne fait pas le métre », les whiskies foncés sont encore aujourd'hui synonymes de maturité, de richesse et d'authenticité. Pro-



du whisky, est alors devenu une réelle préoccupation. La couleur d'un whisky est fonction du nombre d'années passées en fût. C'est au cours des trois premières années que l'interaction entre le bois et l'alcool est la plus intense. Au-delà de cette période l'influence du chêne diminue au fil des ans. Depuis l'essor des blends scotch au XIX^e siècle, la couleur est parfois ajoutée par ajout de caramel alimentaire. Les maisons d'embouteillage utilisent, en application de la législation, un

fluent du débat actuel sur la coloration et de la tendance au « tout naturel », certaines maisons ont récemment développé des gammes de whiskies « blancs » ou non colorés. L'absence quasi totale de couleur est devenue, soudainement, un critère de qualité. En réalité, elle témoigne surtout de l'utilisation de fûts jeunes en tant que phénomène lié à un usage répété. En aucun cas elle ne saurait garantir la qualité d'un whisky.

Dilution, coloration, filtration

Dilution, coloration, filtration

La dilution

Cette étape consiste à ramener le degré naturel d'un whisky, souvent parfois plus de 60% vol., à un degré minimum de 40% vol., la force minimale en dessous de laquelle un alcool de grain ne peut être légalement appelé whisky. La dilution concerne la majorité des whiskies disponibles sur le marché. Elle s'effectue par adjonction progressive d'eau déminéralisée qui provoque la précipitation et la concentration des acides gras présents dans l'alcool. En plus de cette réaction chimique, la dilution modifie les arômes et les saveurs du whisky.

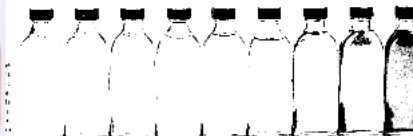
La coloration

Dans les années 1960, dans un contexte de forte croissance, les blenders ont peu à peu remplacé le verre transparent des bouteilles de whisky par du verre teinté. L'impact visuel de leurs facettes, en particulier la couleur et la limpidité

Longtemps ignorées des consommateurs, les pratiques de dilution, de coloration et de filtration sont devenues des sujets particulièrement sensibles. Depuis quelques années, le débat entre, d'une part, les consommateurs et, d'autre part, les distillateurs et les embouteilleurs indépendants, amène à se justifier sur l'intérêt de telles pratiques.

canon, de type 1750, employé pour la coloration de boissons fortement alcoolisées. Le caramel est introduit après la dilution du whisky, juste avant sa filtration.

Si l'assemblage de dizaines de fûts entre eux (blends) permet d'obtenir une constante en termes d'arômes et de goûts, la couleur résultant de l'assemblage n'est pas nécessairement la même d'un batch à un autre. Le changement de teinte d'un whisky, si léger soit-il, tend à provoquer une « crise de confiance » chez les consommateurs peu avertis. Pour beaucoup, la variation de couleur implique nécessairement un changement de qualité, ce qui est loin d'être toujours le cas. Un whisky de couleur ambrée et couleur flûte plus facilement l'œil du consommateur qu'un whisky arborant une teinte légèrement dorée. Même si « la couleur ne fait pas le métre », les whiskies foncés sont encore aujourd'hui synonymes de maturité, de richesse et d'authenticité. Pro-



du whisky, est alors devenu une réelle préoccupation. La couleur d'un whisky est fonction du nombre d'années passées en fût. C'est au cours des trois premières années que l'interaction entre le bois et l'alcool est la plus intense. Au-delà de cette période l'influence du chêne diminue au fil des ans. Depuis l'essor des blends scotch au XIX^e siècle, la couleur est parfois ajoutée par ajout de caramel alimentaire. Les maisons d'embouteillage utilisent, en application de la législation, un

fluent du débat actuel sur la coloration et de la tendance au « tout naturel », certaines maisons ont récemment développé des gammes de whiskies « blancs » ou non colorés. L'absence quasi totale de couleur est devenue, soudainement, un critère de qualité. En réalité, elle témoigne surtout de l'utilisation de fûts jeunes en tant que phénomène lié à un usage répété. En aucun cas elle ne saurait garantir la qualité d'un whisky.

6.4 Image rotation

If the image we want to treat is not straight, the program cannot recognize the characters, therefore the program won't work. That is why we wanted to implement a rotation function in our program. The image rotation was a delicate part to handle. It is not an exact science so we had to choose which way to handle the problem. First we had to determine the angle of rotation and then write a rotate function with the angle as a parameter. I will explain my work

6.4.1 Rotate with a given angle

In order to rotate the image we had to apply a rotation matrix to it. So we consider our image as a matrix of pixel which we multiplied by the rotation matrix, outputting new values for each pixel. To do so, we traversed each pixel and applied the following formula to it(_x and _y being the coordinates of the matrix after rotation) :

$$\begin{cases} _x = \cos(\text{angle}) \times x - \sin(\text{angle}) \times y \\ _y = \sin(\text{angle}) \times x + \cos(\text{angle}) \times y \end{cases}$$

6.4.2 Determining the angle

This step was, in my opinion, the most complicated one in the image processing. The information we found was a bit fuzzy but with many sources we managed to pull out a method that sounded interesting : The Hough transform. Basically it consists in drawing lines on the picture and the line crossing the most black pixel, is the one that shows the degree of inclination of the picture and will help us for the deduction of the angle of rotation.

I had trouble with that last part. I did my best but I did not manage to furnish a functional code in time for the deadline.

7 Block, lines and characters detection

To be able to pass letters to our neural network, we first had to recognize them on images. In order to do this, we use the Run Length Smoothing Algorithm (RLSA). Characters recognition is done in 3 steps :

- Blocks detection
- Lines detection
- Characters detection from lines

7.1 How RLSA works

This algorithm consists of two steps. At first, we loop through every line and column. For each one, we create a binary sequence array : a cell's value is set to 1 if the corresponding pixel is black, and it is set to 0 if the pixel is white. Then, we will change the value of white pixels depending to a fixed threshold.

- White pixels become black if the number of adjacent white pixels is less or equal to the fixed threshold.
- Black pixels stay black.

By doing this on every line, and then every column, we will obtain two different bitmaps (one representing the list of modified lines, and one for columns). It is mandatory because spacing in documents may be different on lines and on columns. Then, we merge the two bitmaps in a logical AND operation : if the two pixels are set to 1, then the final pixel will be 1, and 0 if that's not the case.

Once this is done, we should obtain something similar to the following image :

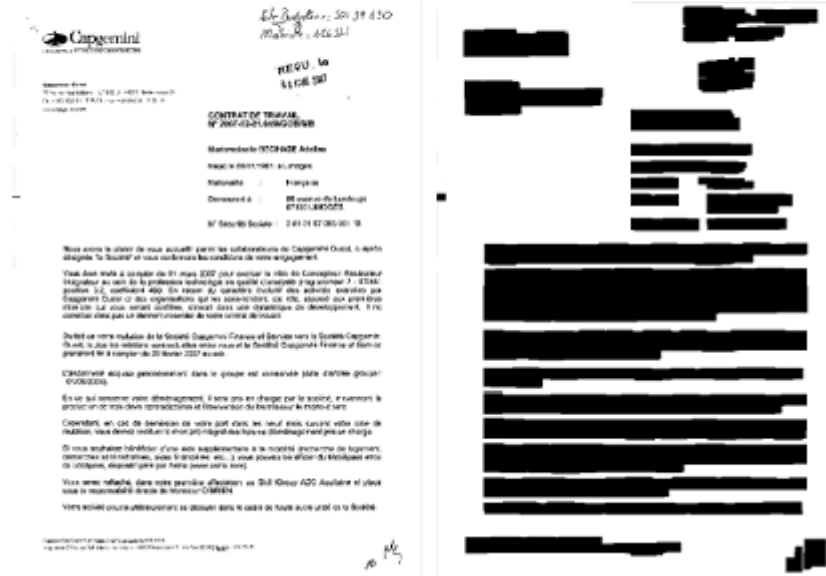


FIGURE 2 – Block Detection

This algorithm is used for both block and lines detection. To do so, it is only necessary to adapt the threshold. As its value increases, it will recognize bigger and bigger text blocks. The algorithm is applied on a copy of the initial image, so that it is not yet modified.

The next step is to draw boxes around lines in the original image, by finding the black rectangles on the copy. We must loop through each pixel of the copy, and check if their color is set to black or white. If it is black, it means that we found a rectangle. We set the current pixel as white, to be sure that it will not be detected twice, and then we recursively check the surrounding pixels to see if they are black too. By doing this, we will be able to get back the width and the height of a black rectangle, and then store those values inside an array. We are now able to draw boxes around lines of text. By drawing those boxes in a different color than black and white, it is easier to get them back if necessary.



FIGURE 3 – Line detection

Once we have properly detected lines, it is important to extract each character from them, and store them in an array.

As the data about lines is stored in an array (the origin, width and height), we can directly find lines. The thing to do, is to check for a white separation between two letters (a white column, or diagonal if the text is not well disposed). It will make smaller boxes, and we will store their coordinates, width and height in another array. Then, those boxes will be given to our neural network.

7.2 Our implementation of RLSA

7.2.1 Line detection

Our implementation of RLSA is not exactly the same as the standard one.

At first, our block detection was not as accurate as it should have been. The first threshold I used were quite small, and some letters were not completely black : for example, with an "S", only the lower part was blackened. To fix this, I modified a little bit the X-axis and Y-axis traversal. I used pretty big threshold for horizontal traversing (almost the width of the picture), so that I was sure to get an entire line with all characters (including last letters, and

curved ones). For vertical traversing, the threshold is similar to the old one : not too large, to be sure that it won't detect the blank between all the text lines, but large enough to color a whole character.

Thanks to those fixes, I was able to get the text line by line. Also, characters were all recognized : points, commas, points on "i"...

7.2.2 Characters detection

Once I had all the text lines, character detection was not that complicated. Basically, I had to go through the image vertically to detect black parts (which correspond to lines). When I find a line, I check every column of the original image, corresponding to the line I detected, to see if it is composed of white pixels only, or not. If it is, then it means that there are no letters here. If it is not, then there is a character, so we set the column of pixel to blue.

A black line



Original text at this
emplacement:

Lorem ipsum dolor sit amet,

Set to blue columns
that are not white only:



FIGURE 4 – How character detection works

We process the whole image using this method, to finally get back all the characters :

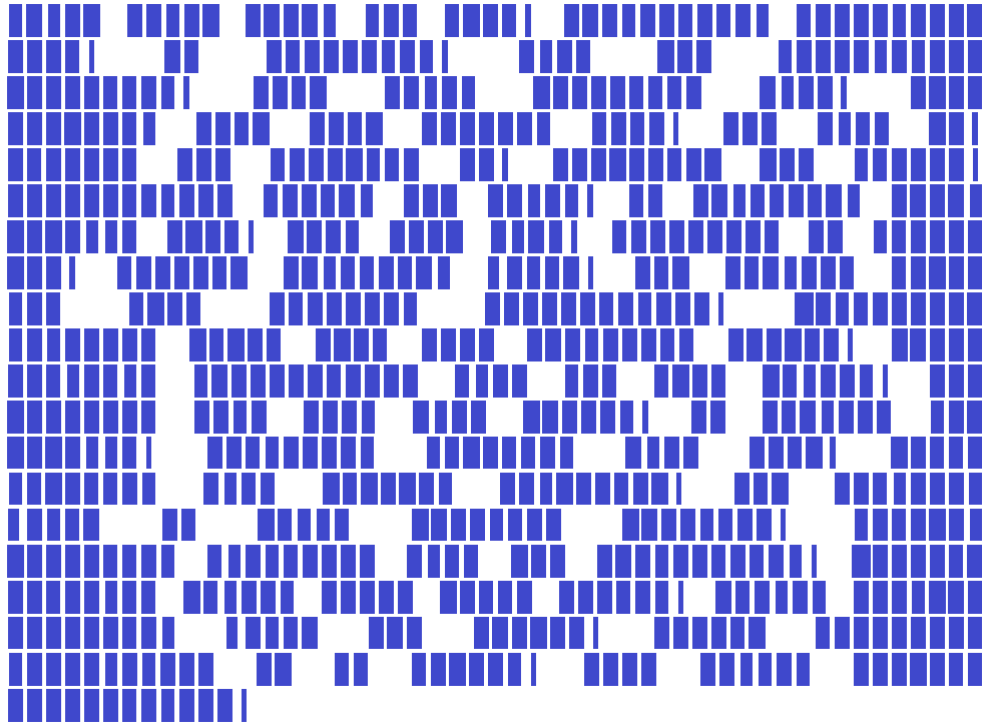


FIGURE 5 – Final Character Detection Example

7.2.3 Characters Extraction

This is the final step of text detection. While making all the blue rectangles on characters emplacement, I call another function which returns the character associated to the corresponding rectangle. It is returned as a `SDL_Surface`.

Each `SDL_Surface` must be processed, to fit a predefined size : 28x28 pixels. To do so, I had to create a new surface, and then copy pixels from the original surface to the new one (after processing). In general, the surface was a little bit bigger than this size, because the thresholds i chose for RLSA left some margin around characters during detection. Basically, it was more about removing those margins, so the surface would fit. Also, the letter had to be centered : if the surface was smaller than 28x28, I had to compute the difference between 28 and the size of the surface, and divide it by two to get the margins I had to left on each side.

In order to store all the `SDL_Surface`s, I use a simple queue. I only need

two basic functions : push into the queue, and clear the queue. All the characters are pushed into the queue in the good order, so the text-reconstruction will be easier. Once the text is recognized, the queue is cleared and we are ready to detect another image's text.



FIGURE 6 – Surface Output

8 Neural Network

First sketches for the XOR

The Neural Network is for us the most challenging part. In order to take it from the right angle, we thought reading existing code was a good solution. Indeed, there is a huge step between understanding theory and implementing it yourself. Studying different versions of neural network lead us to consider 2 options. We could either build a static or a dynamic implementation. At last we did both.

On the one hand, the first one presented many advantages. It was a good starting point to understand back propagation algorithm. The code was short and simple. Didn't needed us to get out of our knowledge comfort zone. It mainly cleared out the 'feed forward' principle as the 'feed backward'. The loops of computation and propagation became more and more explicit in our minds. Besides, the Xor didn't necessarily need a complex structure to be 'learned' by a computer.

On the other hand, it was too simple for a more complex goal. As the finish idea is to -try to - create a Character Optical Recognizer, we needed to have something easier to modify and manipulate for the next steps. In this state of mind, we chose to instantiate our BPNN -Back Propagation Neural Network- as a C structure. It's characteristics are it's number of inputs, Sigmoid in hidden layer and of outputs.

Overall, both are quite similar in idea - but not in structure. Basically, at the end of a propagation - input to output or otherwise - we measure the difference between the actual value and the expected one. Then we go back on our propagation, modifying the weights and threshold according to this delta. After many iterations of learning - epochs - the delta stabilized after having decreased. Then the final output should be really close to the expected one.

Even though this seems satisfying - as it works with both implementation - it only provides one output value. Yet we would win a lot in precision having

a vector made of n units - n the number of different outputs.

Example

Here, we can see the output of the application of those algorithm on Xor inputs and outputs.

```
Iteration 99100 Error :7936.734943
Iteration 99200 Error :7938.499269
Iteration 99300 Error :7940.261946
Iteration 99400 Error :7942.022979
Iteration 99500 Error :7943.782370
Iteration 99600 Error :7945.540123
Iteration 99700 Error :7947.296240
Iteration 99800 Error :7949.050726
Iteration 99900 Error :7950.803583
0.000000      0.000000      0.028118 (Should be 0.000000)
0.000000      1.000000      0.976049 (Should be 1.000000)
1.000000      0.000000      0.980623 (Should be 1.000000)
1.000000      1.000000      0.013308 (Should be 0.000000)
```

FIGURE 7 – Xor outputs

It's a choice we made to put a high number of epochs. Like this we can see how slower the errors are increasing. The higher the number of iterations is, the more precise the result get. However, we can notice that the two outputs supposed to be '1' are different. This comes from the computations inside.

```
Iteration 100   Error :51.275854
Iteration 200   Error :102.596559
Iteration 300   Error :154.974604
Iteration 400   Error :208.350639
Iteration 500   Error :262.617328
Iteration 600   Error :317.676560
Iteration 700   Error :373.440967
```

FIGURE 8 – Error progression

Indeed, one set of input and output of the xor table is a training set. The second one already benefits from the previous computations and modification. Therefore, it is more precise.

Final BPNN model

For character recognition, we chose to build a neural network with 184 inputs, 15 sigmoid in the hidden layer and 62 outputs. It can be easily explain as we have image of dimension 28 x 28. Therefore there is one output per pixel of it. Then, the output is composed of 62 sigmoid decomposed as follow : 10 for digits, 26 for lower characters, and 26 for upper. It outputs a vector of dimension 62, with contains percentage of resemblance with each characters.

Indeed the higher percentage at index i will correspond to the char value $(i + k)$ in ascii table, with k a coefficient to compose a vector uniform with no gaps - as the characters are not all aside in ascii table.

Moreover, there were no need to change the back propagation part. We kept the sigmoid function as an activation function. We noticed that the similarity was quite hight for the desired character. More, all the other outputs were tending to zero.

However, it was not decreasing significantly enough on it's associated capital letter. To resolve this, we tried to implement a Softmax function on the output layer. The theory was good. It should have increased the good coefficient while lowering all the others significantly. Yet, we faced some unexpected results : all the outputs were flattened around the same value - 40 - 50 %.

So, rather than trying to get too deeply into neural networks black magic, we choose to carry on with what we had.

It was then time to train our neural network properly.

At first, we we resized all of the image inputs to the right dimensions - which is for us 28 x 28. Then, we binarize the image and transform it in a vector of zeros and ones. It will be the input of our neural network. The target, the expected output values, are created with a sets of image character label - the name of the folder for us.

- but with imagination you may guess the original text.

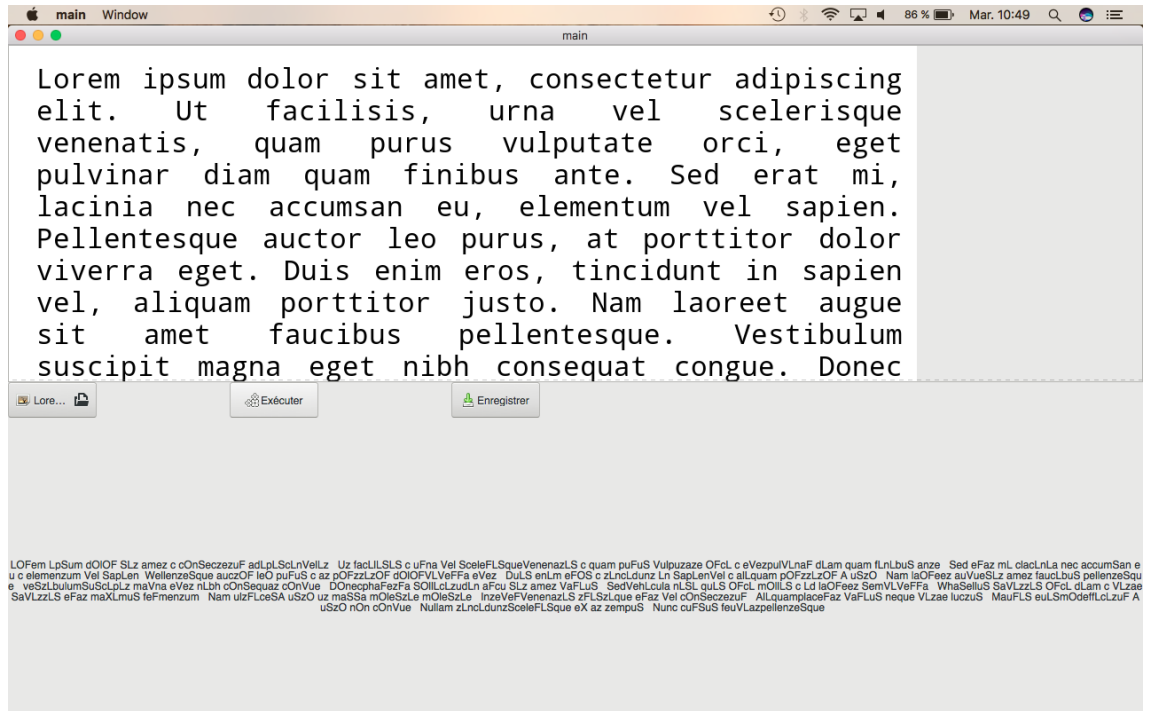


FIGURE 10 – Recognized Text

*'LOFem LpSum dOlOF SIz amez c cOnSeczezuF adLpLScLnVelLz Uz fa-
cLlLSLS c uFna Vel SceleFLSqueVenenazLS c quam puFuS Vulpuzaze OFcL
c eVezpulVLnaF dLam quam fLnLbuS anze Sed eFaz mL clacLnLa nec ac-
cumSan eu c elemenzum Vel SapLen WellenzeSque auczOF leO puFuS c az
pOFzzLzOF dOlOFVLVeFFa eVez DuLS enLm eFOS c zLncLdunz Ln Sa-
pLenVel c alLquam pOFzzLzOF A uSzO Nam laOFeez auVueSLz amez faucL-
buS pellenzeSque veSzLbulumSuScLpLz maVna eVez nLbh cOnSequaz cOnVue
DOnecphaFezFa SOllLcLzudLn aFcu SLz amez VaFLuS SedVehLcula nLSL
quLS OFcL mOllLS c Ld laOFeez SemVLVeFFa WhaSelluS SaVLzzLS OFcL
dLam c VLzaeSaVLzzLS eFaz maXLMuS feFmenzum Nam ulzFLceSA uSzO uz
maSSa mOleSzLe mOleSzLe InzeVeFVenenazLS zFLSzLque eFaz Vel cOnSec-
zezuF AlLquamplaceFaz VaFLuS neque VLzae luczuS MauFLS euLSmOdef-
fLcLzuF A uSzO nOn cOnVue Nullam zLncLdunzSceleFLSque eX az zempuS
Nunc cuFSuS feuVLazpellenzeSque'*

9 Graphical User Interface

Once the biggest part of our program was written, we started working on the graphical user interface. Indeed, even if a program works, if there is no way to display it, it is useless (not everyone can use a terminal unfortunately). None of us had ever done an interface so we were all new to the process.

9.1 GTK+

GTK+ is a cross-platform widget toolkit for creating graphical user interfaces. We chose it because it is one of the most commonly-used Interface maker and for the many opportunities it offered, plus, we were advised to work on it. We started by documenting ourselves, we looked for the best way to implement it and we found GTK+.

Notions of Widget

Before starting studying the creation of graphical interface in GTK+, it is necessary to know that the graphical objects of GTK+ are called widgets (Window Gadget). A widget is in fact a structure defining the properties of an object associated with a wide range of functions to manipulate these objects. Here, the term "object" is to be taken literally, but also in the sense of Object Oriented Programming (OOP). Indeed and, although GTK+ is written in C, it introduces the notion of inheritance and GTK+ widgets follow a well drawn hierarchy. So all graphic objects inherit the properties and functions of a base widget called GtkWidget. Thus, the widget allowing a window (GtkWindow) has of course its own functions, but thanks to the inheritance it also benefits from the functions of the other widgets from which it derives.

9.2 Glade

And looking furthermore into how it worked we found out about Glade. Glade is a "timesaver software", it is basically a way to create the interface by drag and dropping (if we roughly minimize the concept).

The perks of Glade are that it is very intuitive and a preview of the final result is easily accessible. So even without having any knowledge of the software, it is manageable to fiddle with it. On the other hand there is not very much documentation on glade. So linking the code with the interface was a bit of a challenge but we managed to do it correctly. The interface we created is quite simple, this is what we wanted. We wanted something clear to use without any useless overload.

This is how the basic Interface looks like when we just launch the program and when we load the image :

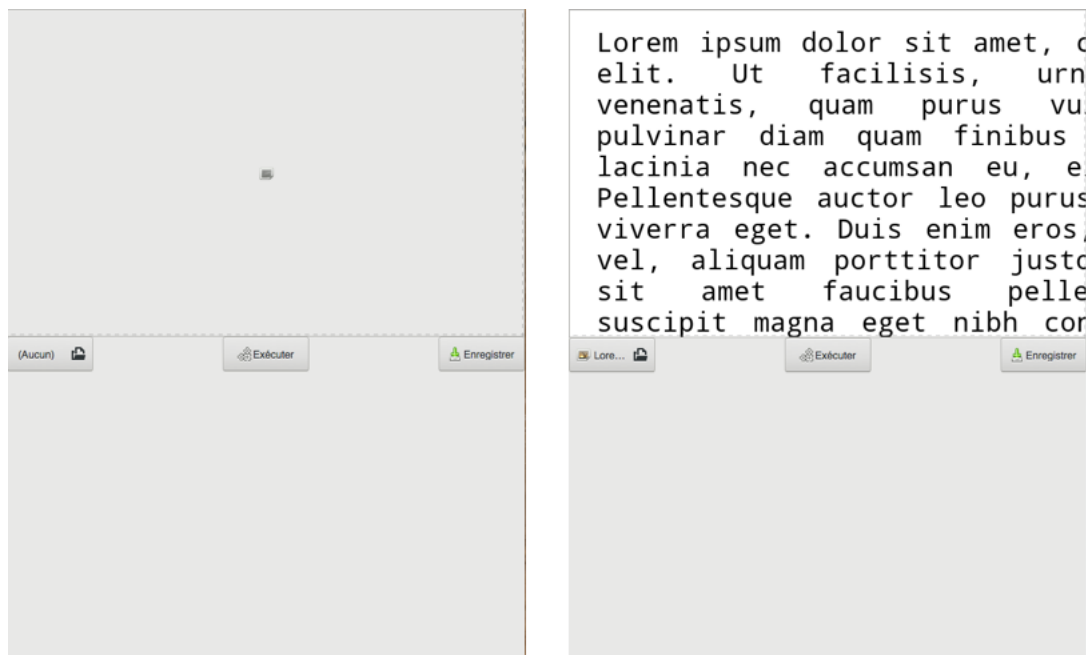


FIGURE 11 – Load of an image

The buttons seen above in figure 12 are the GtkWidgets mentioned earlier.

- the first button calls the browse function, allows to select an image and displays it above.
- the second button calls the binarize function than the segmentation function that is the input to the neural network, so this button basically calls our entire code and displays it in the blank space below (it is apparent in figure 11)
- the last one creates a txt file and stores the freshly generated text in it.

10 Difficulties Encountered

10.1 Guillaume's

One of the first issue I faced was setting up my computer. It is not that evident using Windows OS, since I had to install Ubuntu Bash, but unfortunately it is not possible to open SDL windows on it. It means that on each compilation, I had to save our images as BMP files, transfer them in folder that were directly accessible from Windows OS, and then open the images. Also, I first installed SDL2, and then met some compilation issues, as the environment we have in school is set up with SDL1.2.

Concerning image cutting (block and lines detection), I had some problems with the rectangles we have to make around text. The fact is that you can't do a proper one, since some letters are bigger than the others, and the "I" is in two parts. Due to this, I had a lot of inaccurate rectangles, and in some places numerous little rectangles for a same letter. In order to fix it, I had to change the method I was using to detect characters, and rebuild new functions almost from scratch.

And then came the issues with character detection. At first, we had this constraint of fixed-size character (28x28). Then, each time I was trying to extract my characters from the original image, I just got back a queue of completely-white surfaces.

10.2 Claire's

Working on the training and recognition - globally the neural network - took me a lot of time and energy, and so, time was my worst problem. I needed lot of it for research and debugging. As well, sometimes not working for one day gave me a new look on my work. It was difficult to manage to do 'nothing' when you are really trying to make your program work. Also, I have found like this an answer just after the deadline submission, it was so frustrating!

10.3 Sophie's

The main difficulty I faced was the same as Guillaume, concerning the SDL library compatibility. Indeed the second version removed some units from the library so when we put our code together we had trouble with this.

Concerning the image processing part I had some small problems arising from my level in C, but I easily overcame them since I got better throughout (with a little practice).

However, the major problem I faced was on the rotation It was very frustrating for me because I know that with a bit more time I could have done it. At least it taught me that managing my work properly is primordial.

10.4 Emanuele's

The main issues that i found were just technical. I was not able to use git and because of this I always had to work with someone of the group. Also additional documentation for GLADE and GTK in c language it's really rare and the official documentation sometimes can't answer properly to some practical stuff, making me lose a lot of time to solve banal problems. Because of all this, rarely I was able to work cleanly just on programming and algorithms, spending most of the time trying to find a machine where i could work and reading tons of tutorials just to create basic stuffs.

11 Definitions

Backpropagation Method for computing the gradient of the case-wise error function with respect to the weights for a feedforward network. Hence, a backprop network is a feedforward network trained by backpropagation. For batch training, standard backprop usually converges to a local minimum, if one exists.

Luminance Luminance is the apparent brightness. It is how bright an object appears to the human eye when looking at it from a particular angle of view. For example, luminance is used to characterize the brightness of video displays.

Neural network According to Haykin (1994), p. 2 :

A neural network is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects :

- Knowledge is acquired by the network through a learning process.
- Interneuron connection strengths known as synaptic weights - here Sigmoid - are used to store the knowledge.

Training The weights of connections are adjusted on the basis of data. In other words, NNs "learn" from examples, as children learn to distinguish dogs from cats based on examples of dogs and cats. If trained carefully, NNs may exhibit some capability for generalization beyond the training data, that is, to produce approximately correct results for new cases that were not used for training.

12 Conclusion

In order to have a proper OCR, we needed basic image processing, text extraction of a document and basic neural network. It might seem easy but all of it was new to us so it's not without pride that we present our very own software. We are aware it's not fully functional, but we feel like it's a great step for us.

At first, this project was a bit scary. We had much to do on a unknown subject, and this, in a short laps of time. We did our best in four months to embrace the project fully. In spite of the classwork, the exams and personal projects we invested time on this project and it was worth it.

Appendix

Table des figures

1	Recap chart	12
2	Block Detection	17
3	Line detection	18
4	How character detection works	19
5	Final Character Detection Example	20
6	Surface Output	21
7	Xor outputs	23
8	Error progression	23
9	Example of output	25
10	Recognized Text	26
11	Load of an image	28

Webography

- [1] <http://neuralnetworksanddeeplearning.com>, *Theory and explanation on BPNN*
- [2] <http://homepages.inf.ed.ac.uk/rbf/BOOKS/PHILLIPS/cips2ed.pdf>, *Guide image processing in C*
- [3] <ftp://ftp.sas.com/pub/neural/FAQ.html>, *NN FAQ*
- [4] <https://www.youtube.com/watch?v=kNPGXgzxoHw>, *Video Xor neural network*
- [5] <http://www.cs.cmu.edu/afs/cs.cmu.edu/project/ai-repository/ai/areas/neural/cns/0.htm>, *CNS NN source code*
- [6] <http://adaptiveart.eecs.umich.edu/2011/wp-content/uploads/2011/09/The-pocket-handbook.pdf>, *The handbook of image processing algorithm*
- [7] <https://www.git-tower.com/blog/git-cheat-sheet/>, *Git cheat sheet*
- [8] <http://www.tf3dm.com>, *map origin*
- [9] <https://crblpocr.blogspot.fr/2007/06/run-length-smoothing-algorithm-rlsa.html>, *Explanations on RLSA Algorithm*

Table of contents

1	Acknowledgments	1
2	Introduction	3
3	What is an OCR program ?	4
4	The Members of the Team	5
4.1	Guillaume Morin	5
4.2	Sophie Starck	6
4.3	Claire Helme-Guizon	7
4.4	Emanuele Zamattio	8
5	Task distribution	9
5.1	For First Presentation	9
5.2	For Second Presentation	11
5.3	Progression Overview	12
6	Image Processing	13
6.1	The SDL library	13
6.2	GreyScale	13
6.3	Binarization	14
6.4	Image rotation	15
6.4.1	Rotate with a given angle	15
6.4.2	Determining the angle	15
7	Block, lines and characters detection	16
7.1	How RLSA works	16
7.2	Our implementation of RLSA	18
7.2.1	Line detection	18
7.2.2	Characters detection	19
7.2.3	Characters Extraction	20
8	Neural Network	22

9	Graphical User Interface	27
9.1	GTK+	27
9.2	Glade	27
10	Difficulties Encountered	29
10.1	Guillaume's	29
10.2	Claire's	29
10.3	Sophie's	30
10.4	Emanuele's	30
11	Definitions	31
12	Conclusion	32