Computer Project File
SPE–2022

OCR
2dn Report
By 4pm

Cloé LACOMBE
Victor DUTTO
Pierrick MADE

Wednesday, December 05, 2018

English Version

Contents

1 Introduction

This year, as spe-students in Epita we have to do a three months computer project in groups of four (or three). This project is allowing us to put into practice all knowledge acquired in lectures, tutorials and practicals. Furthermore, it is also improving our personal skills which could not be put into practice during courses.

Our group is named 4pm and we have been working on an optical character recognition software. Also called an OCR the aim of this software is to detect the text of an image and translate it to a text. Hence, it allows the user to take a photo of a document and get the contained text.

This Report will present in details the progress carried out since the beginning of this project. A first section will give detailed information about the concept of the project and will present our group. Then, a section will give the progression and organization of this project followed by the last section with all the details for each task. In other words, there will be the distribution of the tasks, their progression over time and details about them. These sections will try to answer those questions : What was expected in this part of the project and what has been achieved? How this part evolved and how was it done? Finally, the last section will be a conclusion for this report, including personal feedback and a general conclusion.

1.1 Object of the study

This project allows us to improve many of our skills. As a group or individually, it gives us many useful tools. This three months project is for us a second real project which was really rewarding.

Here are the main skills that this project gave us :

- Working as a group :
    - Management
    - Team-working
    - Planning
    - Organization of a project
    - Git for a cooperative coding work
- Programming knowledge :
    - C
    - neural networks
    - image processing
    - LaTex for documents and Reports
    - git as a version-control system
    - Enhancing autonomous learning

1.2 State of the art

A quick presentation of OCR through history

The very first OCR-Optical Character Recognition was created in 1929, which was made of a light pointing toward a word if it was in its memory. During about 50 years, OCR concept evolved using several implementation, until that, in 1978, the reading machine, a software created by Ray Kurzweil and commercialized by Stevie Wonder was published.

Now most, that is to say all, of the implementations follow this pattern of process:

* Pre process - The image is corrected to be clearer. For instance, it is in this stage that the image is turned out in a black and white one.
* Segmentation - The image is separated in chunks.
* Recognition - It is the were the program recognizes the characters from the image
* Post process - Checks if the results seems coherent. For instance, it checks that all the words of the texts belongs to the wanted language.
* - Generation - Generation of the outgoing format

1.3 Presentation of 4pm

4pm is a group made out of 3 EPITA students.

- Cloe Lacombe - 19 years
  Quite familiar with several Linux distributions
  Very efficient under pressure.
  Very motivated when it comes to achieving personal projects.
  Interested in pretty much everything related to IT and sushies

- Pierrick Made - 19 years
  Likes to explore the Linux features, understand how libraries should be used and other very useful skills.
  Very curious and self-learner
  His vim configuration has unexpected but welcomed features.

- Victor Dutto - 18 years
  Interested in researches about AI.
  Interested in abstract modelisations.
  Efficient in reading and having a grasp of how systems works, less in implementing.

2 Organization of this project

To be able to keep a project organized and be able to respect deadlines, the tasks division could not be neglect. Therefore, this section will present in details the different tasks of our project, their distribution and their progression over time.

2.1 Resources aspect

For this project we use mainly free and open-sources softwares or available tools as:

- Overleaf (LaTex editor)
- VIM (file editor)
- Git (Version-control system)
- Glade (for the graphical user interface)
- Internet (For documentations and everything else)

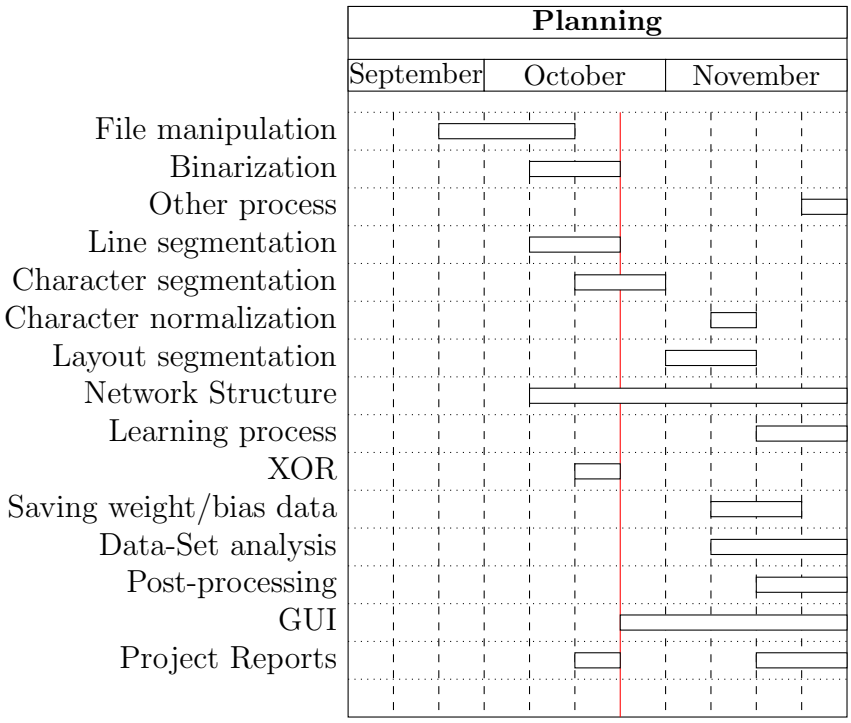
2.2 Tasks Distribution

For a project of this scale, the division of the work was really important. Indeed, without doing it, everyone would have worked on the same thing, and the project could not have evolved like that. This distribution was done at the beginning of the project taking into account our wishes and our skills. However, the issue we came across at this time was our knowledge of group projects and skills in C. Indeed, it meant that our distribution could not be perfect. Thereby, it's why this distribution evolved during the project depending on our skills and the necessities of the time. The following Table is here to present you the actual division that we used.

Tasks	Member in charge	His substitute
File manipulation	Cloé	All
- Load Image	Cloé	All
- Display Image	Cloé	All
- Manipulate pixels	Cloé	All
- Save Image	Cloé	All
Pre-processing	Pierrick	Cloé
- Otsu Binarization	Pierrick	Cloé
- Other process	Pierrick	Cloé
Character segmentation	Pierrick	All
- Line segmentation	Pierrick	All
- Character segmentation	Pierrick	All
- Character normalization	Pierrick	Victor
Character Recognition	Cloé & Victor	
- Basic implementation	Cloé & Victor	
- Network Structure	Cloé & Victor	
- Learning process	Cloé & Victor	
- XOR	Cloé & Victor	
- Saving weight/bias data	Cloé & Victor	
- Data-Set analysis	Cloé & Victor	
Post-processing	Pierrick	Victor
- Spell-checking	Pierrick	Cloé
- Text reconstruction	Pierrick	Victor
GUI	Pierrick	All
- Design	Pierrick	All
- Linking and merge	Pierrick	All
Code organization	Pierrick	Victor
Reports	Pierrick & Cloé & Victor	

2.3 Progression

To give an overview of the progression of tasks over time we created a ganttchart containing steps. On This chart the red line is showing the date of this first submit and the end of this chart corresponds to the end of the project. On this chart we can clearly see a higher amount of work on the end of the schedule resulting from our really low level in C at the beginning of the project.



3 Parts of the OCR

An OCR, in the State of the Art presented above, follows a pattern which is rarely contested, since the programs following this pattern are having quite good results. The implementation of our OCR follows this design paradigm, which implies that our OCR will mainly focus on typed texts and recent handwritten texts, and won't be able to deal with more specific texts like Qumran manuscripts and old copies. To do so, we will use several steps.

3.1 File manipulation

The first step in a OCR is to be able to manipulate files: it is a very necessary part whitout wich we would not be able to do much. This is the reason why we began to work on this part soon during the project.

3.1.1 Load an image from a file

Loading an image is pretty important in an OCR, since the point of the project is to create a capable of "reading" the text on a picture. Thus, we need to be able to load the picture to read.
Loading a picture is actually just about calling a function in the SDL library, the function Load_image. This function need to include the "SDL\SDL_image.h"

3.1.2 Saving image & text file

Just like loading an image, saving an image is just about calling a function from the SDL library, the SDL_SaveBMP, wich requires to include "SDL\SDL.h".

Since we also needed to be able to write into files, mostly to save the weight/biases for the neural network and to save the output of the OCR, we chose to use the function fprint, which write the string passed as an argument in the file given.

3.2 Pre-processing

After loading the image, the first step of all ocr is the pre-processing. This step is here to process the image and make the character detection & recognition easier. As a matter of fact, this section will consist on steps like grayscale, binarization, de-skew, and so on to facilitate the further detections.

3.2.1 Grayscale

The first thing to do in an OCR is removing the colors of the image. This means turning any colorful picture into grayscale. Setting a colorful image as grayscale is necessary for further use : as we will discuss in part 3.2.3, the binarization, wich is the step where the picture set in black and white.

The usual way to turn an image into grayscale is basic math. It is based upon a formula, wich calculate the luminescence of each pixel. The luminescence of a pixel is actually the quantity of light in the colors of the pixel, wich also means its level of grey. Convenient right? The formula to calculate the luminescence of a pixel is as follow:

$$luminescence = 0.21R + 0.72G + 0.07B$$

Where R represents the red color scale, G represents the green color scale and B represents the blue color scale.

Now that we know how to have the light of a pixel, let's get back to work.. To convert an image into greyscale, we have to do it pixel by pixel. Luckily, pictures are represented by matrices, where each element represents a pixel, so to get to every pixel, we just have to iterate througt the matrice. Then, for every pixel¸ we get their RGB value (the value of each of their red, blue and green component). Since we used the SDL library for the picture processing, we had a function that allowed us to get theses values, the SDL_GetRGB function, wich takes the pixel as parameter. To get the pixel at position i,j in the pixel matrice, we were given the function get_pixel. With the RGB value of each pixel, we could apply the luminescence formula, to finaly change each RGB value for the luminescence. Again, thanks to the SDL library, we already had a function to apply these values to a pixel, the SDL_MapRGB function. The final step was to put back the pixel with its new RGB value in place. For that, we were also given the function needed, put_pixel.

3.2.2 Binarization

We needed to binarize the colors of our pictures, because the step of segmentation needs the image to be in black and white, thus the matrix to have only two different values (0 and 255). We tried a first method of binarization, the basic one : the algorithm goes through the matrice of pixel and check the value of the red component (we only need to check this one since the image is in greyscale, thus the R,G and B values are the same). Then, if the value is above 127, the r, g and b values are set to 255 (black), else they are set to 0 (white). For this, i used the SDL functions presented in the last part of the report (part 3.2.1)

3.2.3 Otsu Binarization

The binarization seen above is a basic way of giving a binarized image. However, this method encounters its limits really fast when we start to change the colors of the text or the background of the document. Indeed, giving a black text on dark background will give back an image completely black. It's here that the Otsu method comes in action.

The Otsu method was created by Nobuyuki Otsu and is providing an efficient threshold for binarization. Nobuyuki Otsu has engaged in mathematical fundamental research and its application concerning pattern recognition, image processing, multivariate analysis, artificial intelligence, and neurocomputing. Otsu's method, an image binarization technique, is still a standard technique widely used. This implementation assumes that there is two classes of pixels, foreground and background, it then finds the most optimum threshold to separate those two classes. In our context of text detection, this algorithm is really good as we should have the text as the foreground and the rest as the background.

The main concept of this method is the use of an histogram showing the quantity of pixels of each gray-level in our grayscale image. This histogram is then used to compute the probability of finding a pixel of this gray-level in the image. This probability is used in the following otsu's formula :

$$\sigma_w^2(t) = w_1(t)\,\sigma_1^2(t) + w_2(t)\,\sigma_2^2(t)$$

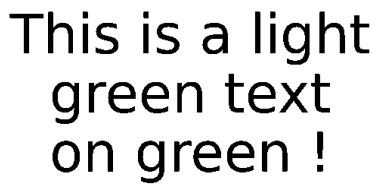
w_i : probabilities of the two classes separated by a threshold t

σ_i^2 : variances of the two classes.

The desired threshold corresponds to the maximum of σ_b^2 which is expressed in terms of class probabilities ω and class means μ. This threshold is then used exactly as for the simple binarization seen above.

The last step is a quick process to turn the text in black (resp. background in white) if the binarization with otsu threshold gave back a white text (resp. black background). This process only use the probabilities of having more white pixels in some parts of the image for black colored text.

(a) Initial image (b) Otsu Binarization



(c) Binarization result

3.2.4 More Pre-processing methods

There is always possible pre-processing to do on input images to allow a larger range of documents. Here is a list of possible pre-processing that could be implemented later or that are relevent to show.

De-skew : This may also be referred to as rotation. This means de-skewing the image to bring it in the right format and right shape. The text should appear horizontal and not tilted in any angle.

Despeckle : Sometimes it happens that the acquired images present dirt consists of isolated points blacks, due to dust or even to electrostatic interference due to the electronic components. The english term used to identify this type of dirt is the word speckle, and the cleaning operation for the identification and removal of these points is defined despeckle.

Remove Noise and Scanning Artefacts : Noise can drastically reduce the overall quality of the OCR process. It can be present in the background or foreground and can result from poor scanning or the poor original quality of the data.

Lines removal : This step would allow us to clean up non-glyph lines and boxes preset in the image. Indeed, those lines can lead to misrecognition.

Brightness and contrast adjustment : To avoid issues with binarization, a strategy of brightness and contrast adjustment can be done.

3.3 Character segmentation

In this section the layout and the segmentation of text will be analyzed. The neural network that we are coding is only able to take a character image as an input, not the whole text. It's why this part is something mandatory in any OCR software. More precisely this part will do the segmentation of the text into blocks, lines, words and then characters. Seen like that we clearly see the recursion process used. As a matter of fact, in a scanned document, we will find the different blocks, in each of those blocks there are lines containing words and finally characters.

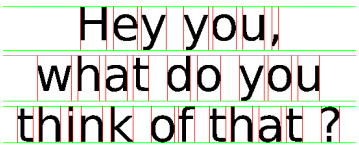
3.3.1 First Implementation - XYcut

At the beginning our aim was to give back something working even if it was not perfect for all ranges of documents. To familiarize with C coding the best solution was to code an XYcut.

This method is based on the assumption that all text-lines of the document will be separated by at least one white pixel horizontal line. Respectively it assumes exactly the same for characters being separated by at least one vertical line.

For this implementation we created a struct named TextLine containing upper and lower bounds of a line. This TextLine also contains a list of Character. Naturally, the second struct is Character which contains the left and right bounds of a character.

Hence, to implement the XYcut on textLines the goal is to iterate through all the horizontal pixel lines of the text-image. If we find lines containing black pixels surrounded by white lines we consider it as a text line. Respectively it is the same with characters but using vertical lines.



(a) XYcut

Unfortunately, this method does not work for a more general case where there can be characters without a clear separation, there can be blocks of texts, and so on. Even if it's not perfect we are happy of this implementation as it works without any bugs. Furthermore, it helped us understand the struct concept and the recursivity of this exercise.

3.3.2 Recursivity and Structure

The segmentation that we are doing is completely recursive. Indeed, our document contains a list of blocks which contains textLines which contains words which contains Characters. All of this structure can be represented as a general tree. Seeing the segmentation this way really helped us for our implementation.

Reconstruction of the text after the segmentation is then really simplified if we see it as a general tree. Indeed, making a pre-order treatment allow us to put back in the good order all the wanted text. By going through the tree we only have to add spaces and newlines in intermediate treatments. Even if we did not implement a general tree, our struct give the same global strucure. We created a struct text which contains a list of struct block wich contains a bounds and list of textLines, Each TextLine contains lines bound, the average charcter width and a list of characters, then our Character contains a matrix representing it (to be passed to the neural network).

3.3.3 RLSA method

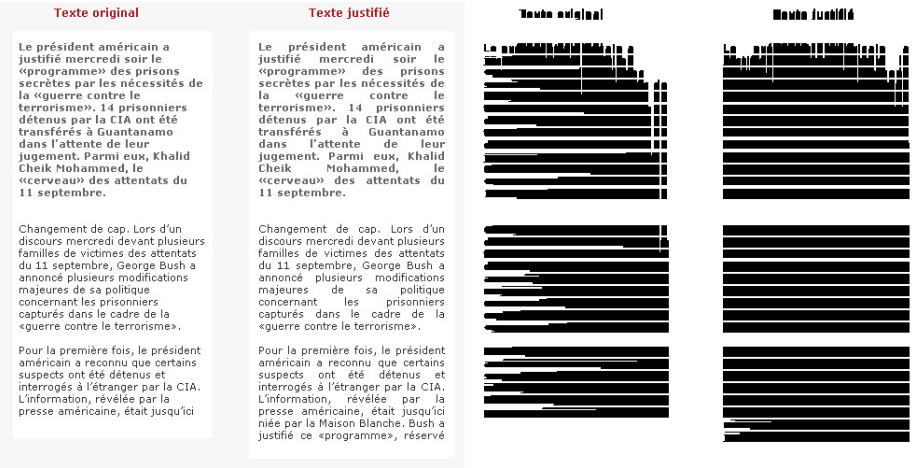
From its entire name "Run Length Smoothing Algorithm" this method is often used in text segmentation for its efficiency. This method allow us to blacken the areas to be detected. Once blackened, they are perfectly isolated from the rest of the image. By the exploitation of the coordinates of these "black zones" we can determine with a good precision which zone of the image we want to process. We thus obtain by analysis of the blackened image the general tree of the elements to be treated.

The method consist on passing through the image twice. Starting for example with the rows and then perform the column processing. This treatment consists in blacking the white pixels that are part of a set of black pixels. To do this, we pass to the function a parameter that will determine

under which maximum interval the number of white pixels in a row must be found so that they go black. For example, with a parameter set to 4, the sequence S is transformed in sequence C.

$$S = 00010000010001001000000011000$$
$$C = 11110000011111111000000011111$$

The two bit-maps created from row processing and column processing are then combined in a logical AND operation.



(a) expected RLSA result

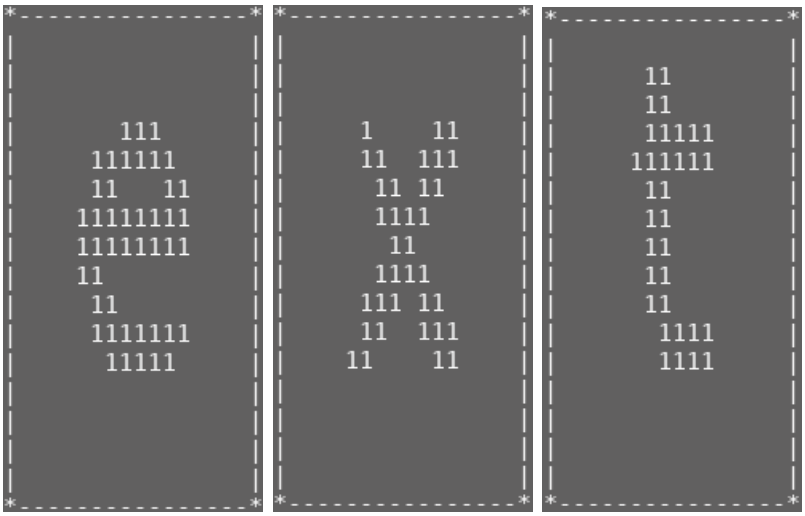
We did implement the rlsa method, this was done as a completion of our XYcut implementation. This method would have allow us to do a nice layout segmentation that could have different types of containers as images or different paragraphs. Unfortunately, we did not manage to finish this segmentation and concentrate ourselves to other more important issues.

3.3.4 Character normalization

The last step of our character detection is about pre-processing characters to a nice format for the neural network. Indeed, we need to give to the neural network a small squared image for each character detected in the document.

To give back this small format image, a work was done to change the scale of the image, and change the aspect ratio without altering the character. This work was done according to the data-set that we found to train our neural-network.

By convention we decided to get matrix of 16x16 for the input of our neural-network. The aim was to convert all found character to this size. The parts of this work was done by putting the segmented character at the middle of a larger matrix of size n x n (with 16 dividing n). Then a simple interpolation would give us a centered character in a 16x16 matrix.

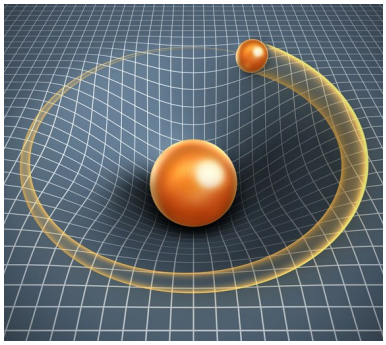


(a) 16x16 char from database and text segmentation

3.4 Character Recognition with Neural network

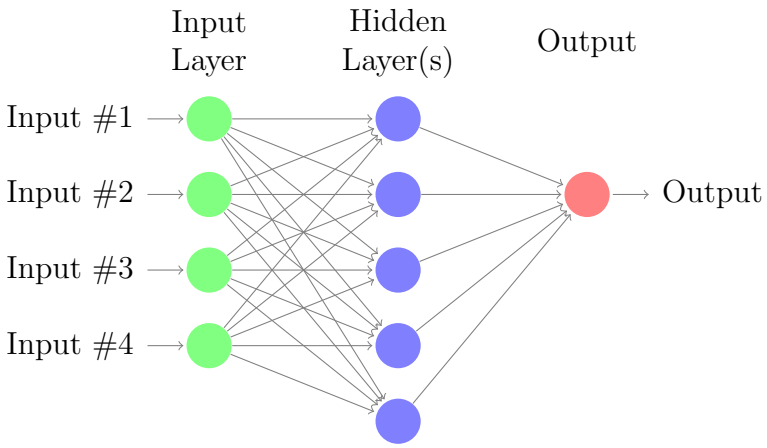
A NeuralNetwork is just a structure using nodes, to compute some output with respect to some inputs. Then, it tries to evaluate how to be as close as possible of a wanted value, which can be taught or not, within the function it is representing. Thus, it is important to specify that a NeuralNetwork will try to find the local minimum. We can do an analogy with topography to do so: The NN is trying to reach the lowest point, thus, it'll check the slope

from its location in several directions, and make a step in the way where the slope will be the greatest. Thus, it is very case-sensitive, and the NN is quite bad to find the minimum of a function. It does just find very quickly a local minimum.



3.4.1 Network Structure

To have some good result, we can use the Neural Network structure. A Neural Network is made of several layers. An Input Layer, which contain the Input given to the structure. Hidden Layers, that are handled by the structure in itself. The Output Layer, which contains the given output. The graphical representation would be :



3.4.2 A bit of formulas

Roughly speaking, at a given layer, we have several inputs of the form:

$$\lambda_{i=1}^{n}$$

are all the inputs from the previous layer.
They are all associated to a given weight w_i, that is a component of the neural net in itself. Each link between node i and node j has its weight. And each node has it own bias, that comes after the sum of the product of weights and inputs. We denote all the lambdas as a vector a, of n dimensions.

$$\lambda\left(a\right) = \sum_{i=1}^{n}(w_i\lambda_i) + b$$

For a given input from a previous layer l, we would wish a value between l and l+1. And we would also need a slightly more precise re partition, and to make critical the impact of the variation of a given wi. The value of the output of a layer should be a value is between 0 (any correlation with an element at a given state) and 1 (the neural net is sure that the element is valid). +1 will be computed using a sigmoid function such that :

$$\sigma\left(a\right) = \frac{1}{1 + e^{(-)*\sum_{i=1}^{n}(w_i\lambda_i)+b}}$$

The goal of this method is to learn, using some training inputs, how to deal with all possible inputs. To do such, we can feed it with inputs, knowing if the neural net should accept them or not. Once the output is known, the network "backpropagates" the error, correcting at each hidden layer how the node dealt with it.
To correct the error, the network use gradient descent method,the principle is: In a layer, given an input from the previous layer, we can compute the error, and modify a parameter, make the error lower.

In this implementation, the gradient was badly computed, making it impossible for the neural network to correct itself correctly. Yet, only time was

lacking to correct it, and thus, the behaviour of a neural net is well understood.

3.4.3 Yet a lot to learn

The formula to compute the error is quite long, but here, I computed the partial derivative of a directly, as a 1 dimensional vector. Thus, the neural network cannot correct its errors, even if it feeds itself correctly, the backprogation of the error cannot be efficient.

The backpropagation is what make a neural network convenient and viable as an extractor of informations from a database.
Indeed, the output of a node is a given value, thus, we want to see how its error margin, which is embodied by the cost function

$$C\left(w,b\right)=\frac{1}{2n}\sum_{x=1}^{p}\left\{y(x)-\sigma(w,b)\right\}^{2}$$

n being the number of training inputs in the set when used with the stochastic method.
w being one of the weight (of course the functions here are taking in account all weights)
b being the bias
y(x) being the output wanted at this stage
but sigmoid being the one the node returned.
The cost symbolizes the error margin, how far the result is from the expected value. Thus, to make the neural network efficient, we want to correct this error margin, and thus to find the minimum of the cost function. Thus, we want to compute its gradient. The gradient of a function is the vector made of all the partial derivatives of the several parameters of the function. We want to know the director coefficient of the tangent created by a partial derivative, to see how can we modify the parameters in a way that the function will be as near as possible of its local minimum.

But the neural network is really efficient because the gradient of the Cost behaves as some basic multiplications, for a given weight w(i)

$$\frac{\delta C}{w_i} = 2y(x) - a_i$$

If y(x) is registered at the beginning of the training, which is the case during supervised training, the only variable if a(i), which can be registered. Thus, it is really easy to compute a gradient vector, the order of complexity is around the length of the layer a(i), which is also the length of the gradient vector. And, in order to have each node error, we can grab a layer vector of errors

$$Vect_L = \left\{ \begin{array}{c|} \delta_1^l \\ . \\ . \\ \delta_i^l \end{array} \right\}$$

Once this vector is stored, we can compute each element of the next layer using the next formulas

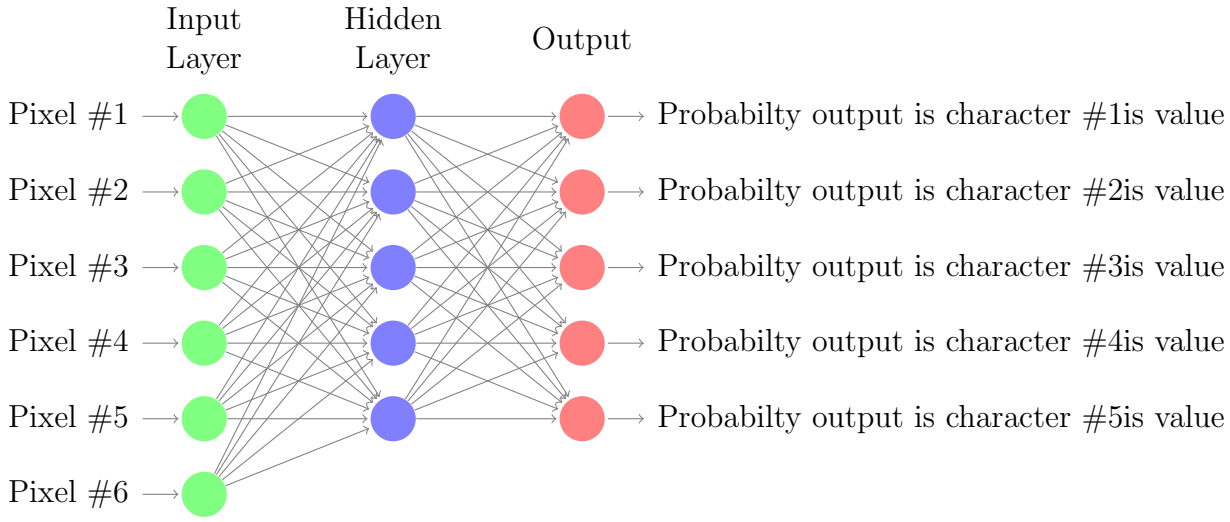
$$\delta_j^L = \sum_k w_{kj}^l \delta_k^l \sigma'(z_i^L)$$

With L = l - 1 w the weight associated to delta and sigma prime the derivative of the sigmoid function defined previously.
Therefore we can notice that the error can be either positive or negative, as the sum of weights can be of both signs, the sigmoid prime is always positive and the error can be of both signs.
Thus, it would be the error of a given node in the hidden layer. The neuralnetwork structure is so efficient since its complexity order is only around n times the number of weights! Even if the gradient stochastic searching method does exist since the 50's, the usage of neuralnetworks as a powerful tool of data managing is only around a decade old! But that's already used pretty much everywhere because of its efficiency, so we wanted to implement it as optimized as possible.

Once all the gradients are computed, each weight and bias of each node is recomputed to fit the best its new goal, its new value to reach.

3.4.4 Using Neural nets as character recognition

When used with images, the neural net checks the value in each 'pixel' of it, and then compare the output with respect to the expected output. In our case, the image of a given letter is a 16*16 matrix. Thus, the input layer is embodied by an array whose length is equal to 256. On the neural we wanted to implement, we wanted it to have an input layer of length 256, a dozen of neurons at its hidden layer, and around 70 outputs for all lower and upper letters, numbers, and special characters like . ; , ? !



3.4.5 How did we implemented it

To implement the neural network as code, we could have used structures, which is a tool in c that allows a programmer to create a type of variables and give it a given set of data of different type, for instance a structure could be a node for a linked list that would have its own value and a pointer to the next node of the list.
For our neural net, we chose not to use any structure. We only used arrays, for instance:

- A synapse is represented as an array of weights, where its bias is the value located at index zero of the array.

- A layer is represented as an array of synapse, that is a matrix of weights.

- The network itself is an array of arrays of arrays, thus a matrix of layers, that is a matrix of matrices of weights.

So, we implemented the network as an array of arrays of arrays of weights.
But for the backpropagation, we still needed to have in memory other variables, like the outputs of each layers and the error of each layer.
To do so, we chose to create even more array, in order to store these information we need later.
We used one array and two matrices:

- First, we used one array of output for each hidden layer. Even if there is only one hidden layer in the OCR network, we coded it so it could adapt to networks with more hidden layers. So, this is a matrices containing one for the outputs of each node in a hidden layer, plus the array of the general inputs of the network.
The reason why we created this array is because, for the backpropagation, we needed to have memorized all the outputs of the hidden layers, which are inputs for other layers and also have memorized the general input of the network.

- Next, we created one array for the general output of the network, because obviously, the general output is again needed during backpropagation. This general output is needed to calculate the error of the output layer, and again it is needed to calculate the error of the hidden layer.

- Finally, we needed an array carrying the errors of each node of each layer, that is the error for each node of the network.
This array was necessary for the backpropagation, again: the error represent how much a neuron has to add to each his weights to have an output that is closer to the one we expect.

The last thing we needed was to be able to access the size of the total network, as well as the size of each layer (that is, the number of synapse in the layer), which can be different from one layer to another, and finally the number of weights in each synapse, that is the size of the synapse, that is equal for the synapse of each layer but can differe from a layer to another.

For all those variable, we chose to use global variables initialized as constant, because we could need them in any function, and in functions that would not call each other. Because of theses facts, we thought that the best and easier way to have an easy access to those variables from every function in our program was to define them as global.

3.4.6 Our neural network successes and failures

As you have seen, our neural network has, in a way, a recursive implementation. Some critical subfunctions had to behave perfectly in order to make the neural network efficient.
That is not the case, thus we know its behaviour, where it does behaves as expected, where it does not, and to which extent it does not.
Basically, when testing out our network, we saw it was not learning with a random input set. The input set had to be made out of the same element to make the neural net able to recognize this element.

For instance, if we tested it with two inputs and correct it to make it behaves as a XOR gate, making its behaviour, steps and correction clearer, we could see several things:

- First of all, the errors were around zero.

- Second, the weights and biases were not updated on the hidden layer. (that is to say, the one from the inputs to the hidden layer nodes)

- Third, all the errors of the associated error matrix were zeros along the column representing the hidden layer.
- If we expanded the neuralnetwork with another hidden layer, it would have the same errors and corrections as the second one.

From all of the above, we concluded we were calling our error computing of a non-output layer with wrong arguments, and we tried to debug it. Thus, we badly planned the end of the project, and we were running out of time and ressources. Thus, we did not corrected it.

But, we can notice that the output of the network will be influenced by the inputs given and the output expected, the output array will goes closer to 0 for each value if we try to have so, and will do the same with 1.

3.5 Post-processing

After the results given by our neural network, other processes can be done to make those results even better.

3.5.1 Spell-checking

A spellchecking feature was implemented directly in the gui to let the user more controls. Indeed, after some tests and reflections, we considered that automatic spell-checking wasn't a good idea with our codes. Indeed, all you need is an extra space in your word, and spell checking would misspelled the two words. However, an interesting post-processing feature is the upper-case and lower-case check. Indeed, after the character recognition, we could normalize characters if an uppercase letter was found at the middle of a word.

3.6 Makefile, Compilation & Dependencies

A makefile is a file (by default named "Makefile") containing a set of directives used by a make build automation tool to generate a target/goal. This file is something really important to avoid issues with an undetermined number of compilation made by hand. In our case even if this file is not that big, here is an overview of its content.

- Default compiler and linker program : gcc
- Main compilation options : -Wall -Wextra -Werror -std=c99 -O3
  Here, the options Wall, Wextra and Werror are here to specify that we want to process and show every warnings as errors. c99 is the C version used for this project. Then O3 is the abbreviation for optimization 3.
- ‘pkg-config –cflags –libs sdl‘ -lSDL_image
  This line is here to link the SDL library used in this project.
- ‘pkg-config –cflags –libs gtk+-3.0‘
  As the previous one, this line is here to link the GTK librairy used for the gui.
- ‘pkg-config –cflags –libs gspell-1‘
  gspell is used for integrated spellchecking in our gui.
- "-lm" is here to link the math.h librairy used by the neural network
- "all" & "clean" : Those two options are mandatory in our Makefile. It allows the user to make the whole project only by typing "make all". In our implementation, this command-line create all needed dependencies and object files but also the main executable file "ocr". This file can then be executed using ".\ocr".

In this project we do not use much external libraries. However, some are really helpful especially for image loading and graphical user interface.

- Dependencies : libsdl2-2.0-0 / libsdl2-dev / libsdl2-image-2.0-0 / libsdl2-image-dev / libgtk-3-dev / libgtk3-0 / libgspell-1-1 / libgspell-1-dev

3.7 Graphical user interface

Also called GUI, the Graphical User Interface is a real tool to facilitate the manipulation of a software by an user. Indeed, a gui is a form of user interface that allows users to interact with a software through graphical icons and visual indicators instead of typed commands and text-based interfaces. As our project is linked to image processing, having a gui give the user the possibilities to load, preview images and get text results easily. This interface also gives the user many tools to perform actions like conversion, loading, saving, copy-pasting, and so on.

The implementation of this graphical user interface has been done for this second presentation using GTK+. Also called the GIMP Toolkit, this is a multi-platform toolkit for creating graphical user interfaces. To facilitate its creation we used Glade Interface Designer which is a graphical user interface builder for GTK+.

(a) GTK+

(b) Glade

As it was our first time dealing with a real gui, we spent many hours trying to get and understand documentations of glade and GTK. GTK is a really well known tool used to create gui. But we unfortunately discovered that glade documentations and tutorials weren't as present as for GTK. Furthermore, the last version of glade - 3.22.1 - appeared to be not really stable and we went through many issues. To give an overview of those trouble, when we were loading a glade file all the horizontal margins were reset, the preview wasn't working until a copy-paste was done, some object names were reset, and so on. To overcome those issues many changes were made by hand using vimdiff and two glade files.

To give a nice overview of our gui and the work done on it we will first show and explain the different windows, then we will analyze each step and actions that the user can do.

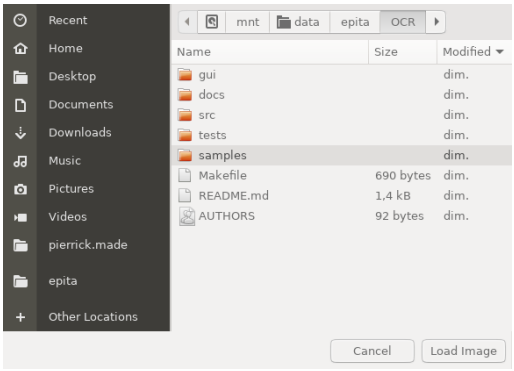
3.7.1 Windows and Graphical Part

Main Window : The first and main window of our GUI is composed of everything useful for the user, it links every actions and windows. Thus, it was the first window we designed and created. The design of this window is really simple, it contains three big parts, the image, the buttons and the result text. The last part is the top-bar menu which gives access to more features like image zooming, spellchecker or some help.

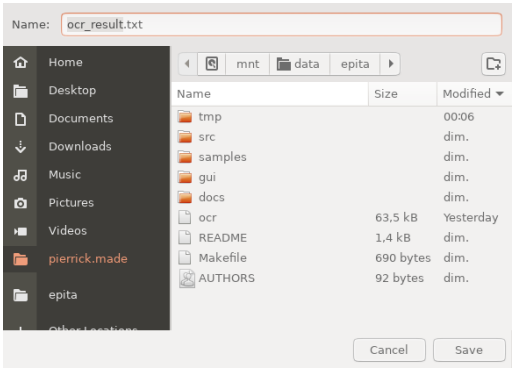
(a) Main Window

About Window : This is not a big window, indeed it contains only some text about us and a link to our github page (github.com/4pm-nomnom/OCR). This was the first page implemented to discover features of glade and gtk. It allowed us to implement our first buttons, GtkImage, box layout and GtkTextView. To discover glade we used prognotes.net and its article gtk-3-c-code-hello-world-tutorial-using-glade-3. It was one of the best website we found to link glade, gtk and C coding. If you want to see our "about window" you can go to our github page linked above.

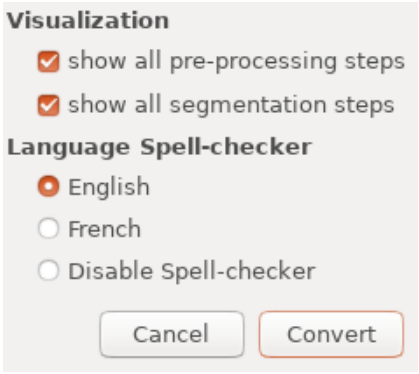
File chooser and saving : Those dialog windows are really useful for the interaction of the user. The user needs to select an image to be analyze by our OCR. These windows are basics file chooser dialog, which we can find on all operating systems. In gtk an object called GtkFileChooser allowed us to create those two windows. Some work was then done to link these file chooser to the main image and to the text to save. One issue we encountered while coding them was to allow the double click and the enter key to select a file. We tried many things, and finally discovered that we just had to make the open button as default.



(a) File opening



(b) Text Saving

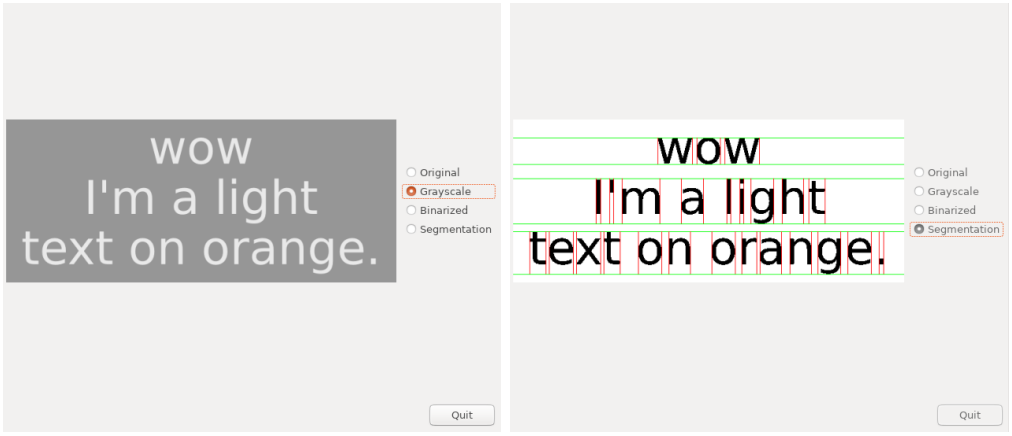


(a) Advanced conversion dialog

Advanced window : The function of this dialog window is to give more options and features to our conversion. This window is opened when the user call the "convert" button with the option "advanced" toggled. Then, when

the user have defined options needed, he can launch the conversion with those parameters. This dialog window have a modular layout allowing us to put more features as the ocr evolve ! For the time, it contains the possibilities to see all processing steps and an option to change the spellchecker. When it will evolve, it could contain the options for cropping, selections, rotation, manual pre-processing and so on.

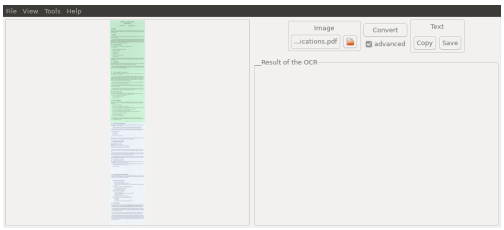
Show steps : In this window the user can see all the processing steps of our OCR, it includes the pre-processing and the segmentation steps. We saw the toggle buttons to active this window in the advanced dialog! When the user will click on "convert" in the advanced window, the result of the ocr will be written and at the same time this window showing steps will be opened.



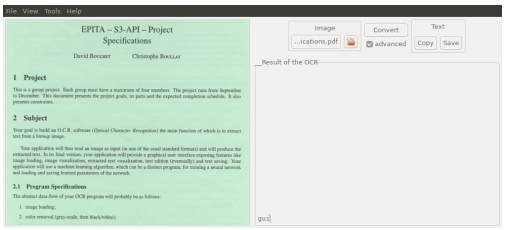
(a) grayscale steps window (b) segmentation steps window

Responsive architecture : Our gui adapts itself to the size of the window asked by the user! If the user want to use a full screen or half of his screen, boxes containing text and image will be zoomed proportionally to use all available space. This was the most challenging feature of our gui as we wanted something really clean and at the same level of known gui. To achieve this tasks we used a container named GtkBox which resize itself linked to its parent. Then for each object we defined their desired size, their parents and how they could expand vertically and horizontally. We can't show gif or moving object in this report, so download our program to test it!

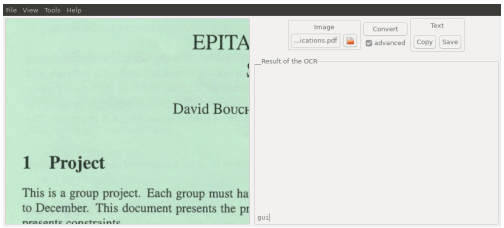
Automatic Image Scaling : According to the responsive behaviour of our gui, the image is scaled automatically when the window size changes or when the user ask it. We implemented four modes for the image view, best-fit, large-fit, normal-size and manual-zoom. All those modes are here to give a more pleasant experience to the user. Indeed, having a big picture displayed on a small screen often need zooming or de-zoom to manipulate it. To implement this feature we based ourselves on the GdkPixbuf object which allow image manipulations. Then we update the main GtkImage object from the modified pixbuf. Secondly, we used the allocation of the parent container which could give us the wanted size. To find this wanted size we used box ratio and image ratio. Finally, to update the image when the main window is resized, we used a signal named size-allocate which is called whenever the main window is resized.



(a) Best-fit



(b) Large-fit



(c) Normal-size

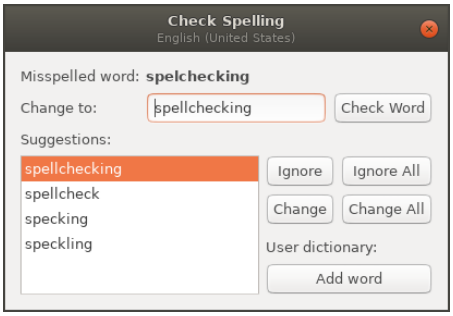
Manual Zoom : In addition to the automatic scaling, the user have the possibility to use "plus" and "minus" keys to zoom and de-zoom the image. The use of those key disable the automatic scaling and let the user select his favorite view. Getting the user input wasn't the easiest part, we had to add a mask to the GtkImage object to enable events on it. Then we had to get the event saying that a key has been pressed and check for the value of this key to do the according action.

3.7.2 User Actions

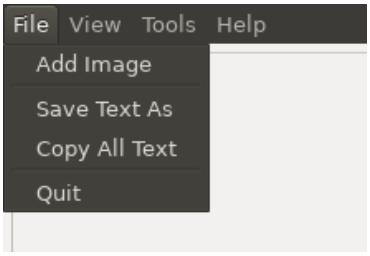
File Loading : To load a file the user needs to open the file chooser dialog using top-bar menu or the main open button (with a folder icon). When the file chooser is opened, the user can obviously open an image to process. If he select another file format, a filtering function will be applied and nothing will be changed. However, we thought that the pdf format would be something great to add in our ocr. After some searches, the easiest implementation to allow pdf was using ImageMagick functions of conversions. Using those functions, we managed to convert the pdf into a large image containing all pages. The main issues of this implementation are its low speed and the max number of pdf page which is 5, else the file is too large to be processed.

File conversion : When the user selected its image, he can click on the "convert" button which will launch our main function convert. The "advanced" toggle button can allow him to set some preferences as seen above in windows part. The result of our convert function is then sent to the textView of the main window which allow the user to see, copy or save the result.

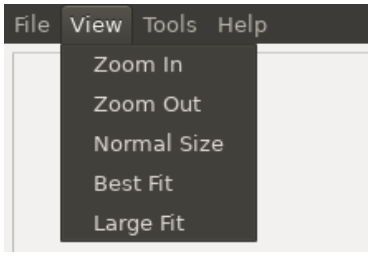
Text Result Processing : The result of the ocr displayed in the TextView object can be copy, modify or saved. The saving of this text is done by using the file chooser again, but with the "save as" option. Before saving it, we found interesting to let the user do some modifications on its text so we gave every right to this text box. Finally, the user have at his disposal an integrated spellchecker which underline in red bad written words and allow him to correct them with a right click. There is also a spellchecker dialog available in the Tools menu from the top-bar. Those spellchecking options are given by the gspell lib which provides a flexible API to add spell checking to a GTK+ application.



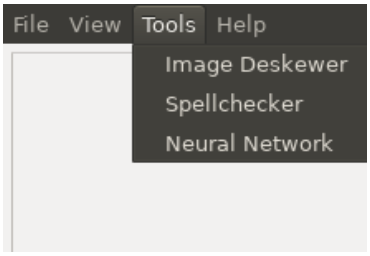
More actions with the top-bar Menu : This bar is a main point in the gui world, it allows the user an access to some features which are not available on main buttons. Giving the user to much buttons in the main window would disturb him from the main aim of the program. But having some more features for advanced users is always appreciated. Our top-bar contains four menus : file, view, tools and help. The file menu is simply linked to the loading and the saving of files as main buttons. The view sub-options allow the user to change the view of the main image, it gives all options presented the image scaling paragraph above. Tools is here to give more tools of processing to the user, including spell-checking, de-skewer or neural-network preferences. Except the spell-checker which works well, other options are not all finished for this presentation but will be implemented later. Finally, the Help menu redirect the user to our github page which contains more information about the using of our program.



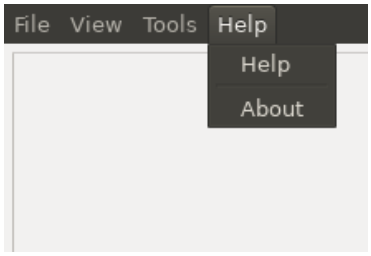
(a) File menu



(b) View menu



(c) Tools menu



(d) Help menu

4 Conclusion

4.1 Personal Conclusions

Victor DUTTO : This project is the first one that I do in this specific configuration.

Indeed, I have the feeling that last year project was more about organization and how to rightfully use softwares whereas this year, we have to code "from scratches" a program. Thus, the qualities that are needed are indeed quite different.
I learned how to use a lot of software, but the main memory i will have of this project will be the end rush that I had to made. Even if I did not succeeded to complete my task, I managed to improve a lot in C and Arch manipulation. The project helped me a lot to integrate the manipulation of several tools, method to think, design, test and finally code what I want to do.

Cloé LACOMBE : I think this project is really interesting because we are asked to code it in C language. Since OCR is based on neural network, and this language is not really suited for this purpose, there is not much documentation for neural networks in c, wich makes the implementation very interesting since we have to do it really by ourselves, we cannot look for code on the internet, wich makes this project a excellent learning tool.

Pierrick MADE : This project really motivates me. Even if its beginning was some kind of hard and we didn't not give the expected result, I managed to do something I'm proud of and I have many ideas for later implementations. Indeed, even this project officially finished, I want to completely finish it and even re-code some parts of it. Thanks to that project, I managed to use the knowledge given to me within courses and tutorials. I also learned many new things linked to the gui creation. I wanted to do something really nice and I'm proud of my results.
In terms of human skills, I discovered how to supervise and organize a group in my last year project. Nowadays, I still have difficulties to lead a group but less to organize their code with git versioning. It is really rewarding to see those results.

4.2 General Conclusion

This project has been, until now, a new way to design and create a program. Firstly, it is the first time we have to do a whole program without using mandatory softwares (we only had to use Git but it does not affect content). In our opinion, this project was about how should we organize ourself in order to deal with long-term enough stuff, that cannot be made in a rush. Globally; we think that; overall; we have to better understand how a team should think about a project, foresee the main problems to come, schedule works sessions, documentation themselves, and others time consuming and resources-requiring activities.
Yet, some features have been implemented, but some mandatory weren't, mostly because of a dangerous way of thinking and organizing the project schedule. Overall, having some rush was useful, so the member which is the least comfortable in C could catch up a little bit on his laxness's, but, not having regular work sessions on the whole semester pushed, in a way, to have too much work to do before the appliance. This is; in our mind, the most important lesson taught by this project.

> " A good programmer is someone who always looks both ways before crossing a one-way street. "
>
> *Doug Linder*

Credits & Sources

Members of this Projects :

Victor DUTTO

Pierrick MADE

Cloé LACOMBE

Thanks to :

ASMs

Mr. Bouchet

Sources :

C Documentation

SDL documentation

GTK documentation

Several libraries in C

neuralnetworksanddeeplearning.com

http://prognotes.net

mattmazur.com

LaTeX Documentation

Youtube's Tutorials