Computer Project File
SPE–2022

OCR

1st Report

By 4pm

Cloé LACOMBE
Victor DUTTO
Pierrick MADE

Wednesday, October 24, 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 first 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 will be really rewarding.

Here are the main skills that this project will give 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)

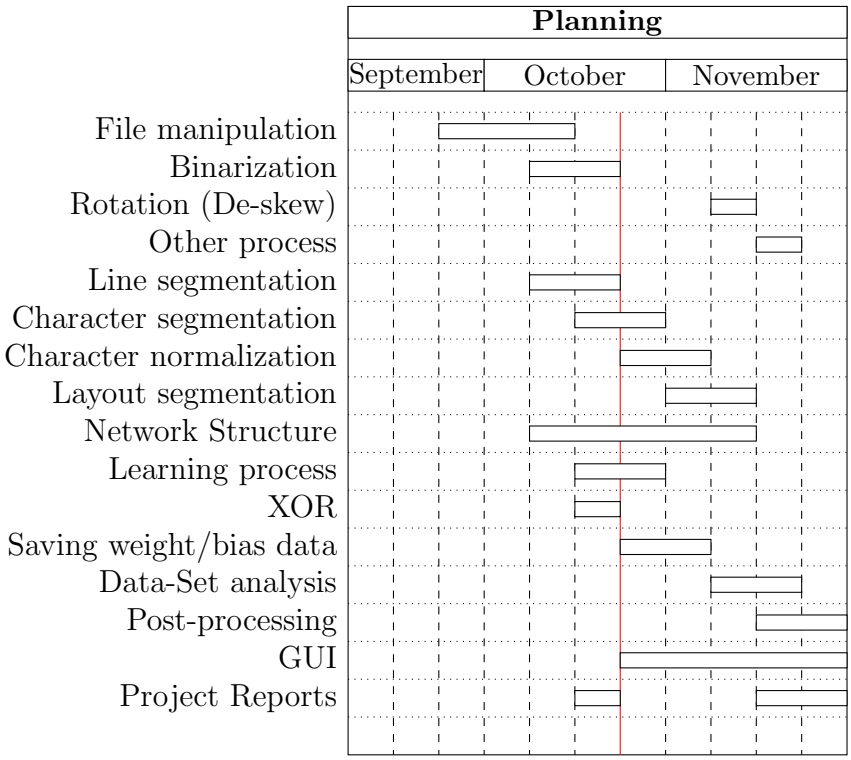
2.2 Tasks Distribution

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	Cloé	Pierrick
- Binarization	Pierrick	Cloé
- Rotation (De-skew)	Cloé	Pierrick
- Other process	Pierrick	Cloé
Character segmentation	Pierrick	All
- Line segmentation	Pierrick	All
- Character segmentation	Pierrick	All
- Character normalization	Pierrick	Victor
Character Recognition	Victor	Pierrick
- Basic implementation	Victor	Cloé
- Network Structure	Victor	Cloé
- Learning process	Victor	Cloé
- XOR	Victor	Cloé
- Saving weight/bias data	Cloé	Victor
- Data-Set analysis	Cloé	Victor
Post-processing	Victor	Pierrick
GUI	Pierrick	Cloé
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 will also need 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 :

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

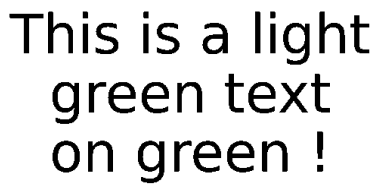
w_i : probabilities of the two classes separated by a threshold t

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 for the next presentation 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.

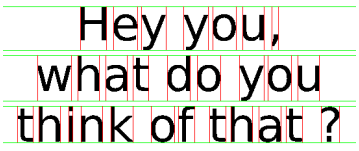
3.3.1 First Implementation - XYcut

For this first defense 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. For this first defense 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

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

Reconstruction of the text after the segmentation will then be really simplified if we see it as a general tree. Indeed, making a pre-order treatment will allow us to put back in the good order all the wanted text. By going through the tree we will only have to add spaces and newlines in intermediate treatments.

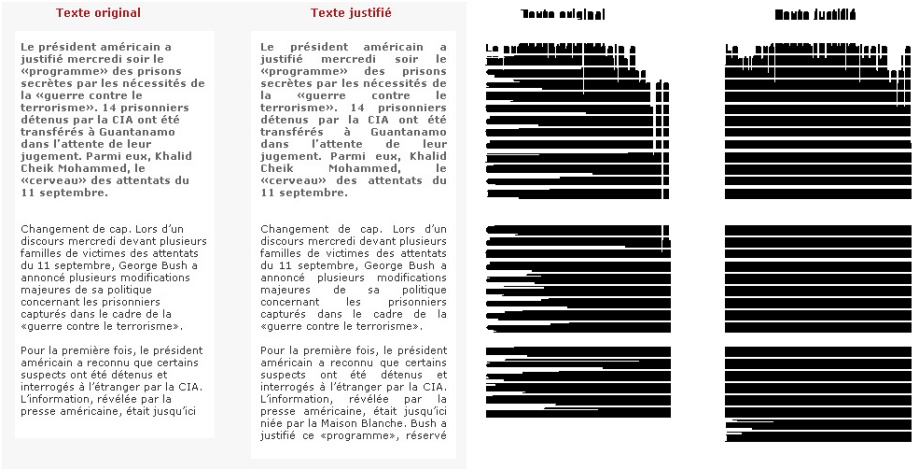
3.3.3 RLSA method

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

For the time being, we did not implement the rlsa method, this will be done as a completion of our XYcut implementation. This method will allow us to do a nice layout segmentation that can have different types of containers as images or different paragraphs.

3.3.4 Character normalization

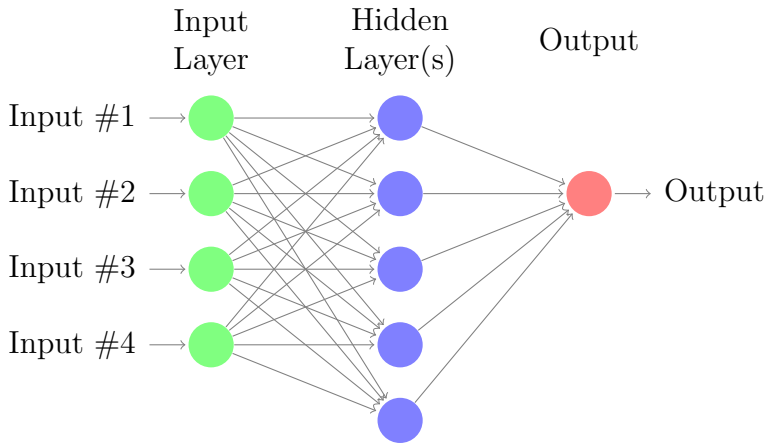
The last step of our character detection will be about pre-processing characters to a nice format for the neural network. Indeed, we will need to give to the neural network a small squared image for each character detected in the document.

To give back this small format image, a work will be done to change the scale of the image, and change the aspect ratio without altering the character.

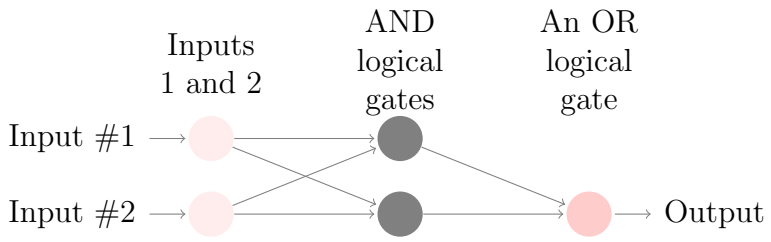
3.4 Character Recognition with neural network

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 Example with : XOR



3.4.3 Learning

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.

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.

3.5 Post-processing

After the results given by our neural network, other processes can be done to make those results even better. For the time this is not our priority, but these could be ideas of implementation at the end of the project.

- Possible checking of the words by using a dictionary or a lexicon - a list of words that are allowed to occur. We could for example use libaspell which is a spellchecker library. It can either be used as a library or as an independent spell checker. Its main feature is that it does a superior job of suggesting possible replacements for a misspelled word than just about any other spell checker out there for the English language.

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.

- Pre-processor options : 'pkg-config –cflags sdl' -MMD
  The first part of this line is linked to the use of the SDL libraries and the second part "-MMD" is linked to dependencies when building.
- 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.
- Libs and path for linker : 'pkg-config –libs sdl' -lm -lSDL_image
  This line is here to link libraries used in this project (SDL, SDL_image and lm for math.h).
- "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 can be really helpful especially for image loading and graphical user interface. For the time we only used SDL and SDL_image to be able to process many types of image file. For the following implementation we planned to use gtk for graphical purposes and aspell for spell-checking in post-processing.

- Dependencies : libgtk-3 / libSDL2 / libaspell

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 will give the user the possibilities to load, preview images and get results easily.

At the time being for this first defense we did not created a real gui. Indeed, the user do not have a window with buttons and options to make selections. The choice of the image is made by passing an argument PATH when calling the program then every steps are done automatically. The only user's interaction is a key-pressing to switch to the next process and display the next image result.

The implementation of a real graphical user interface will be done for the 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 will maybe use Glade Interface Designer which is a graphical user interface builder for GTK+.

(a) GTK+

(b) Glade

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 quite different.
Even more, in this part, I failed to implement properly my neural net, and I personally see this as an opportunity. Because of this failure, I can learn several of my defaults as a developper in the context of a project. Indeed, I should have tested each ones of my function separately to see if they were behaving correctly. But, here, I tested my whole neural network once it was finished, and I was out of time to correct it properly.
Thus, this specific failure will improve me on the procedure of how I think about a project. This project is also making me learning how to use several tools that might get re-used, for instance GitHub, Latex, Vim. Vim and GitHub, especially, are among the most used softwares in the IT professional spheres.

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, I managed to do something I'm proud of and I have many ideas for later implementations. Thanks to that project,I managed to use the knowledge given to me within courses and tutorials.
In terms of human skills, I discovered how to supervise and organize a group in my last year project. Nowadays, I have less difficulties to lead a group and 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). The main challenge for this first defence was to learn how to manipulate new tools and to better understand how a team should think about a project, foresee the main problems to come, schedule works sessions, documentation sessions, and others time consuming and resources-requiring activities.
Thus, the mandatory parts fixed for the first defence are respected, and globally the future of this project seems hopeful. We improved as programmers and as students also, from another perspective, which is more related to the ability to abstract a model and understand it, since we had to read quite a lot of documentation in order to be able to grasp what was asked, how could we implement it and how should we implement it.

> " 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

Bouchet

Sources :

C Documentation

SDL documentation

neuralnetworksanddeeplearning.com

LaTeX Documentation

Youtube's Tutorials