EPITA – S3 – Project Specifications

David Bouchet   Christophe Boullay

1 Project

This is a group project. Each group must have a maximum of four members. The project runs from September to December. This document presents the project goals, its parts and the expected completion schedule. It also presents constraints.

2 Subject

Your goal is build an O.C.R. software (*Optical Character Recognition*) the main function of which is to extract text from a *bitmap* image.

Your application will thus read an image as input (in one of the usual standard formats) and will produce the extracted text. In its final version, your application will provide a graphical user interface exposing features like image loading, image visualization, extracted text visualization, text edition (eventually) and text saving. Your application will use a machine learning algorithm, which can be a distinct program, for training a neural network and loading and saving learned parameters of the network.

2.1 Program Specifications

The abstract data-flow of your OCR program will probably be as follows:

1. image loading;
2. color removal (gray-scale, then black/white);
3. pre-processing;
4. text block detection;
5. character detection;
6. identification of extracted characters;
7. text reconstruction.

The initial stages (loading, color removal, optional pre-processing) are the gate through the core of the OCR. Note that no color information is required for characters recognition and in fact, working in black and white is almost mandatory.

2.2 Image splitting

Dividing the input image into blocks, then characters (eventually splitting blocks into lines) is a mandatory part in order to feed the recognition stage with sub-images corresponding to a single character.

You're strongly advised to organize this stage with a recursive and modular design. In order to be able to extend it for advanced layout (paragraphs, line breaks, multi-columns . . .) All the necessary information will be gathered during this stage for later text reconstruction.

2.3 Character recognition: neural network

Character recognition is the key stage of your OCR. Recognition relies on a training stage where your network will learn how to discriminate characters.

A neural network is tool that can learn a transformation function (non-linear) using training samples: during training, you will provide labeled inputs (image block with the corresponding character) and your network will answer classification probabilities. You will use the error margin to correct the internal stages of the network using gradient retroprogation. Using a sufficient number of samples, you network will, most of the time, provide the correct answers. The seminal outputs for that kind of network is a vector of probabilities for each character. The character with the highest probability will be the recognized character.

In order to implement you neural networks, here are some keywords to look at: ***supervised machine learning***, ***multi-layer perceptron***, ***retroprogation***, ***gradient optimization***, ***AdaBoost***.

The website `http://neuralnetworksanddeeplearning.com/` provides a lot of theoretical and practical information on neural networks and could be useful for your task (the third chapter deals with networks tuning, which is often a difficult part).

There exist many kinds of neural networks, but for this project, you will probably need only a simple MLP with a single hidden layer. Anyway, it can be worth considering a *softmax* activation function (for the last layer) rather than the usual sigmoid. You will find everything you need in previous link.

2.4 Pre-processing

This stage aims at enhancing image quality. This part will allow your program to work with scanned documents. Here is the pre-processing your project must be able to do:

- deskew manually (rotate the image from an angle entered by the user),
- deskew automatically (with skew angle detection and rotation),
- noise canceling (image grain, stains, . . .),
- contrast enhancement . . .

3 Project constraints

Your project must be implemented in the C programming language and must correctly built in the provided environment (Linux). During the defenses, the demonstrations will be done on the school machines. All software installed on school machines is allowed to be used for the project. In addition, you must respect the following rules:

- You must use the `C99` standard.
- You can choose your compiler between `gcc` and `clang`.
- Your code must compile without any warning, using at least the following options: `-Wall -Wextra -std=c99`
- You must provide a `Makefile` with at least the following target: `all` and `clean`.
- You must provide a `README` file as a synthetic documentation.
- You must provide an `AUTHORS` file describing the members of the project (see the format below.)
- All identifiers and comments must be in English.
- Lines must not exceed 80 columns.
- You must remove all trailing white-space.

The `README` file must be a UNIX text file and must be readable in terminal using commands like `cat` or `less`. It can use the *markdown* format. The `AUTHORS` file must contain one line per group member of the form: `* login (LASTNAME Firstname)`

3.1 Version Control and Submissions

For this project, you will have `git` repository usable by all group member. We strongly encourage you to use all the possibilities of version control in `git`.

Before each defense, you must provide the code of your project through your `git` repository.

You will hand in the project using `git`: we will *clone* your repository on the main branch (`master`) on the scheduled day. Only the files presents in you repository when we clone it will be available during the defense.

Your repository must contain:

- The `AUTHORS` file
- The `README` file
- Your source code and its `Makefile`

Your repository must not contain: runable files, external source code (libraries or other projects for example), binary files other than some images used during the defense, report (in any form).

4 Schedule and Deliverables

Here are the deadlines for the project:

September 25, 2020: group submission dead-line

October 26, 2020: first defense (late afternoon & weeknight)

December 14, 2020: final defense (late afternoon & weeknight)

The first defense will be a progress check. Your goal is to demonstrate the current state of your project. Since you're all doing the same project, it won't be necessary to introduce the project nor to rephrase the current document.

For this defense, you will need introduction and conclusion, a demonstration of your project on the expected features described in this document. You will also present how your neural network works. *Take care to present how your own implementation works.* ***Do not give a lecture on neural networks.***

For the final defense, **you must present a complete standalone program**. It is important that this program performs all the expected flow of actions. ***Take care to produce a real program*** *and not a set of unconnected pieces of code*, even if it only works in a restricted way.

4.1 Deliverables for each defense

At the beginning of the defense week, you must submit your code (as stated in the section 3.1) which will be used during the defense. During the defense, you will also provide:

- A report, presenting your work (at least 25 pages for the first defense and 40 pages for the last one): group presentation, distribution of tasks, progress and so on.
- A plan for the defense.

4.2 Project stages and Technical Deliverables

Your project is divided in several parts. A part is said to be completed if your program is able to perform the expected task on significant examples. Of course, the complete evaluation of the part will depend on the overall quality and the robustness with regards to advanced testing.

- **Mandatory parts for the first defense:**
  - image loading and color removal;
  - Blocks, lines and character detection and splitting;
  - a *proof of concept* of your neural network: at least a small implementation able to simulate the *XOR* operator.
- **Parts that must at least have been started for the first defense:**
  - Pre-processing *(at least the noise canceling [image grain, stains, . . .] and the contrast enhancement)*
  - save and load of neural network's weights;
  - set of pictures for learning;
  - File manipulations for result saving.
- **Mandatory parts for the final defense:**
  - All previous mandatory parts;
  - Pre-processing *(everything with the deskew method of your choice)*
  - a complete and working neural network (learning and recognition);
  - extracted text and file saving;
  - A user interface in order to use your program.
- **Optional parts (only considered if mandatory parts are done):**
  - spell-checker;
  - extra: multi-columns, page layout, images embedded in the text.

5 Tips and tricks

This project is designed for a team of four students during 3 months. Some parts, like the neural network, require tuning and sometimes time consuming computations. **It is important to start working on the project as soon as possible, otherwise you may will probably have difficulty to finish the project correctly on schedule.**

You may find it useful to design independent tests for each part. For example, the neural network consumes, as input, images of characters already extracted and scaled. In order to test and train your network, you can constitute a set of images already in the network's input format. This way, you will be able to make progress on this part, even if the other parts are not ready.

During the first defense, you will be asked to demonstrate the expected features, so you should prepare your code in order to demonstrate the different parts of your project easily (stage visualization, independent runs . . .)

There are some common pitfalls in C, where you can waste your time. You will need to take care of a substantial amount of technical details. Develop good work habits in order to avoid wasting time. For memory issues, you could find it useful to use tools like `valgrind` or the `-fanalizer=address` option of `gcc` and `clang`. Also, remember that the most convenient representation of multidimensional arrays (like matrix) use array of single dimension where you just compute shifting explicitly, thereby avoiding conflict between incompatible static and dynamic versions.