A Framework for Using Tesseract to Transcribing Early Modern Texts Having Non-standard Fonts

William Frederick Rial and Sofia Visa
Dept. of Math and Computer Science
The College of Wooster
1189 Beall Ave, Wooster, OH 44691
{wrial16 | svisa}@wooster.edu

ABSTRACT
Here we describe a framework built upon Tesseract optical character recognition software for transcribing old texts having non-standard fonts. Further, we illustrate our software on creating a digital version of two volumes of a 17th century French text. The volumes consist of 808 pages having 84,366 words, and our system initially correctly transcribes 88% of the words. Further, we identify a methodology that will help to correct an additional 1,007 words; this would lead to 89% recognition accuracy.

1. INTRODUCTION
Manual transcription of old manuscripts is labor intensive and error prone. Therefore, we propose a framework that will automate the transcription of old books into digital formats. Our approach builds upon the knowledgebase created with the Early Modern OCR Project (eMOP) [3] in establishing a process by which the machines will read the texts using optical character recognition, or OCR. The eMOP team implemented their own software called Franken+, which is designed to bridge the gap between Aletheia (a parsing tool) and Tesseract (an OCR tool) [6]. This software can be used in our framework, as illustrated later.

Transcribing such books has several merits as pointed out next. Often old manuscripts are available in one or very few copies, and many more students and scholars will have access to an electronic version of these manuscripts. Further, students have difficulties recognizing old fonts in particular when such old manuscripts are used in foreign language classes such as French and Spanish. Finally, digital editions of such texts facilitate annotations and text analysis at a larger scale: for instance Voyant is a tool that allows visualization of some particular words across chapters or even entire volumes, allowing for a more holistic understanding of these texts [8].

We will describe here a three-step approach in transcribing such old font manuscripts, illustrating it on a 17th century old text.

2. A FRAMEWORK FOR USING TESSERACT FOR OPTICAL CHARACTER RECOGNITION
Figure 1 illustrates the three main stages of our proposed framework to transcribing old French texts:

1) remove noise from the image (for our application we have used the ImageMagick free software);
2) parse and recognize text (we have used Aletheia for parsing and Tesseract as the OCR for text extraction);
3) use a post processing software to further fix errors in the text outputted by the OCR.

While in the first two steps freely available software can be used, for the third we recommend a customized module, which incorporates additional external information. For instance, for the transcribing problem described in the next section, we have used our own algorithm implemented in a Python
script as the postprocessing step. Our algorithm is designed to detect and fix errors specific to our own application, and makes use of two French dictionaries and a custom-built error dictionary. As such, this framework can be used in other similar applications with the following changes: a slightly different set of parameters might be needed for the free software used in the first two steps (or even other software can be used); the script from the third stage must be adapted to the specific problem.

Next section illustrates the use of this framework on transcribing old French texts having a non-standard font.

Figure 1. A three-step framework for using Tesseract OCR for transcribing old texts: remove noise, parse and recognize text, and postprocess

3. APPLYING OUR FRAMEWORK TO TRANSCRIBING A 17TH CENTURY MANUSCRIPT

Our framework described above was used for transcribing the second volume of Madeleine de Scudéry's Conversations Sur Divers Sujets (1680). The volume consists of 454 pages, and is available in a digital format (pdf) from the Bibliothèque Nationale de France (5). Below we illustrate all the three steps applied to this volume.

3.1 Removing noise in the tiff image

To remove noisy dots from the images we have used ImageMagick, which is a free software that can be used to manipulate images, including noise removal, color manipulations, geometric transforms, editing, etc. [4]. Before starting the processing, we exported the pdf file into individual tiff images (one image per page). Figure 2 illustrates the third page of this volume before removing noise (left) and after using ImageMagick (right). As it can be seen, only smaller dots of ink were removed because attempting to remove larger spots will alter the text: for instance, all the dots on the “i” letter will also be removed. Therefore, ImageMagick requires fine-tuning for each particular OCR problem; the best parameters for our problem are discussed next.

After investigating several blurring techniques, along with the sharpen command, the most effective square filter for our images was 6x5. While 6x5 appeared to damage the text more than 3x4 for example, Tesseract performed better in the first case. Therefore, the interaction between the first two steps of the framework must be carefully investigated, i.e. an image that visually looks cleaner than another, might perform worse on the OCR step.

Figure 2. Tiff image of page three (left) and its cleaned version outputted by ImageMagick (right).

3.2 Training and Using Tesseract OCR

This is the most laborious stage and encompasses at its turn several steps
including: letter (or glyph) parsing, parameter tuning, and training. Finally, at this stage a “font” is created which is further used for character recognition [5]. Even though Tesseract is a command-line software, here we have used the Franken+ graphical user interface which wraps around Tesseract and greatly helps the user with the steps enumerated above [6].

For parsing the letters we used Aletheia [3], which made original guesses on the letters. Next, Aletheia allowed us to correct several types of errors such as removing characters that were not representative for a given symbol, and adjusting characters we wanted to keep for our final training “font”. For instance, Figure 3 shows on the left how the letters “to” were labeled as the single character “w”; through Aletheia we were able to separate the two glyphs and re-label them correctly. The right side of Figure 3 illustrates the relabeling of the glyph “o” which originally was mislabeled by Aletheia as “n”.

At its turn, Franken+ allows for corrections as well: for instance, Figure 4 shows in red some incorrect or poor representations of the letter “a” which are about to be removed by the user.

Before starting the actual OCR training, Franken+ allowed for other edits and parameter tuning. For instance, it allowed for adjusting the y-axis offset helping in differentiating between lower and upper case “p” (i.e. “P” vs. “p”; or “Y” vs “y”). For our problem, it was sufficient to assign a y-offset of +10 for many descending characters. The two special cases were “p” and apostrophe (“’ ”). Without a large offset of +20, Tesseract could not differentiate between lower and upper case “p”. Upon testing with many different y-offset values for the apostrophe, it became evident that Tesseract always had difficulty discerning apostrophes from commas. Even with an offset as large as -45, Tesseract made no noticeable improvement in separating these two characters.

For training we have used thirteen pages of the volume, i.e. all the letters of these pages were parsed with Aletheia and edited for correctness with Franken+. This allowed for 5-10 good examples for each letter and symbol.

The output of the OCR for the third page of the volume is shown in Figure 5. As it can be seen, the transcribing is not perfect, thus a postprocessing module is further needed.

3.3 OCR postprocessing
As Figure 5 shows there are many incorrect words and symbols, and the postprocessing algorithm fixes some of them. For instance, “qusentre” should be “qu’entre”, and the output of our postprocessing algorithm...
implemented in a Python script fixes it (see Figure 5).

Our postprocessing algorithm is illustrated in Figure 7, and makes use of two French dictionaries obtained from the ARTFL Project at the University of Chicago [1], and an error dictionary built based on the OCR output. Namely, to build this dictionary of errors we visually inspected and analyzed the OCR output of 30 pages and created a dictionary of all wrongly recognized words, together with their correct versions. A snapshot of this dictionary is shown in Fig. 6.

Roughly, our postprocessing algorithm analyzes each word \( w_i \) for correctness in the following way:

- If \( w_i \) is at the end of the line, and in conjunction with the next word \( w_{i+1} \) it forms a correct French expression (i.e. exists in the French dictionary), then the two words are outputted together. Examples of such outputs: “avez-vous” is outputted as “avez-vous”; “ne-gocie” is fixed as “negocie”. The two words are also outputted together if they start with capitals: for instance “Pont-Neuf” will be outputted as “Pont-Neuf”.
- If \( w_i \) is in the middle of the line and it is found in the French dictionary it is outputted unchanged. Examples: “suis”; “mois”; “papier”.

Figure 6: Dictionary of errors: each row shows an error outputted by the OCR and its corresponding correction.
• If \( w_i \) is in the middle of the line and is not found in the French dictionary but it is found in the error dictionary, then its correct replacement from that dictionary is outputted. Example: “rne” is replaced by “me”; “mox” is replaced by “moi”; “LsAmour” is replaced by “L’Amour”.

• If \( w_i \) starts with a capital it is left unchanged. Examples: “Paris”; “Louis”.

• If \( w_i \) is a contraction then an attempt is made to recognize the word as described next. For example, “j’en” is correct but may not be in the dictionary. If the second part of it (here is “en”) is in the dictionary and the first part can be part of a contraction (we have built our own list of ten such characters), then the word is outputted as is; in our example, as “j’en”. However, if the original \( w_i \) was “:’en” then it will be processed by “Is One Word Error” function in the above algorithm.

• If \( w_i \) has a minor error associated with an accent on the letter “e” that is not present in the dictionary of errors, then the One Word Error function tries different accent marks and checks the dictionary. If there is a match with the changed accent mark, then that new word is outputted, otherwise \( w_i \) remains unchanged.

• If \( w_i \) is one word and is not found in the French dictionary or in the dictionary of errors, then we attempt to fix this error as described next. We calculate the sequence matcher ratio \( r \) and the Levenshtein distance \( d \) between \( w_i \) and all French words, and the 3 French words with the highest ratio \( r \) and the minimum distance \( d \) are written in a separate
text file while the system outputs \( w_i \) with error tags (i.e. \(<\text{err}> w_i </\text{err}>\)). The reason \( w_i \) is not replaced with the word having the greatest ratio and lowest distance is explained below.

The sequence matcher \( \text{ratio}(w_1, w_2) \) function is a built-in Python function which compares the two strings \( w_1 \) and \( w_2 \) and returns the similarity between them as a number in the range \([0, 1]\). If \( T \) is the total number of elements in both sequences, and \( M \) is the number of matches, then the ratio \( r \) is defined as shown in eq. (1).

\[
r = 2 \times \frac{M}{T}
\]

(1)

For example, ratio \( r \) for the words \( w_1 = "abcd" \) and \( w_2 = "bcde" \) is 0.75.

The Levenshtein distance between two words \( w_1 \) and \( w_2 \) is defined as the minimum number of changes to transform word \( w_1 \) into \( w_2 \). For instance, “bavoir” can be changed into “savoir” by replacing “b” with “s”.

Hence, the Levenshtein distance between the two words is 1.

For instance, with this approach, “occasion” will be correctly recognized as “occasion”. The \( r \) and \( d \) values for comparing these two words are, \( 14/16 = 0.875 \) and 1, respectively. In contrast, if “occasion” is compared with “location”, the \( r \) and \( d \) values are \( 10/16 = 0.625 \) and 3. There are however, many instances where the word obtained through this method is incorrect. For example, when the word “activite” undergoes this process, we erroneously obtain “actinite” \( (r = 0.875, d = 1) \) as the top match. Therefore, we leave this step to a French speaker to make the final decision.

4. RESULTS

With the algorithm outlined above we transcribed two volumes consisting of 808 pages with a total of 84,366 words. After the OCR and the post-processing steps, 88% of the words are transcribed correctly. However, the post-processing module identifies an additional 1,007 words which are currently labeled as errors, but could be easily fixed with the sequence matcher ratio-based methodology describes in the last step of the module. Namely, we can correct those words by replacing them with the French words closest to them (here a threshold of 0.9 sequence matcher ratio-based similarity or higher was used). This would lead to 89% recognition accuracy. However, in future work a French student will determine how many of these 1,007 words are truly correct.

5. CONCLUSIONS AND FUTURE WORK

Here we illustrate a three-step framework for transcribing old books written in non-standard fonts. Our approach is build around Tesseract and is showcased on a 17th century French text, where it achieved 88% transcription accuracy. As an extension of this work, we will look at improving the postprocessing module as well as quantifying the amount of error it corrects. We will also be applying this system to transcribing five additional French volumes written by the same author. Each volume has a different non-standard font and we plan to compare the performance of our system across these volumes (in terms of training time and quality of resulted text).

6. REFERENCES


[8] Voyant see through text, Voyant Tools is a web-based reading and analysis environment for digital texts, voyant_tools.org. Access date: June, 2015