University of West Bohemia

Faculty of Applied Sciences

Tools for Semi-automatic Preparation of Training Data for OCR

User Manual

Ladislav Lenc, Jiří Martínek, Pavel Král

Pilsen 2019

Chapter 1

Introduction

The presented tools have been developed in order to facilitate preparation of training data for optical character recognition (OCR) systems. Nowadays, most OCR systems utilize neural networks that recognize images of whole text lines and provide its transcription. Recurrent neural networks (RNN) are usually utilized for this task [1]. A great benefit of RNN based approaches is that the segmentation to characters is not necessary. The segmentation free approaches need a significant amount of annotated line images to be able to train the models. The images can be synthesized using available fonts. However, for capturing all aspects of the real data synthetic data are not sufficient. Therefore, some amount of manually annotated data is still necessary. Moreover, the real data must be also used for system evaluation.

The developed tools should serve as simple, one-purpose applications for training data creation. Their main purpose is to create annotations for line based OCR systems, however one of the tools allows also creating character based annotation. It was developed in the context of a project dealing with historical German data. Therefore, there are some special aspects making processing of German texts efficient. The tools can be easily extended for arbitrary languages.

Chapter 2

Character Segmenter

The first proposed tool is used for segmenting text lines into individual characters. It additionally saves the annotated line images as well as the character separator positions. The input of this tool is a set of extracted line images. To be able to obtain an initial set of text lines we utilized Leptonica library [2].

The proposed character segmenter takes a line image and proposes a segmentation to characters. The algorithm is based on projection profiles.

The input image is first inverted and thresholded. Then we calculate the vertical projection profile of the image. This process is illustrated in Figure 2.1.

The white peaks indicate presence of characters. The values lower than a specified threshold are considered to be gaps separating the characters. The proposed segmentation of the example image above is shown in Figure 2.2. This figure shows that several ending character borders are similar to the starting ones, which is depicted with one single separator.

The example shows several segmentation errors that occur typically in letters *m, n* or *u*. Another issue are some pairs of characters that are impossible to split using projection profile method. Examples are *ch*, *tz* and *ck*. The user interface allows manual correction of incorrectly segmented characters.

Figure 2.1: Original text line and its vertical projection profile.

Figure 2.2: Proposed segmentation of an example line image.

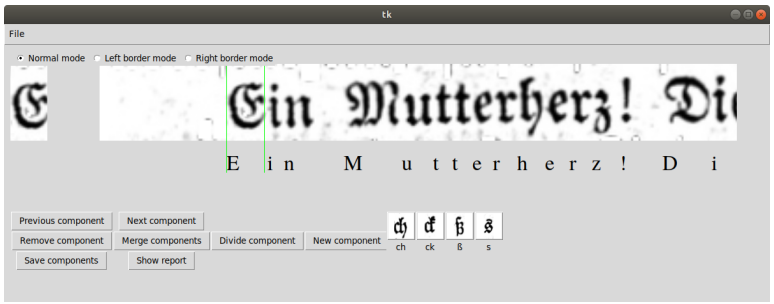

Figure 2.3: Line segmenter GUI.

The tool has options for merging or splitting incorrect segmentations. It is also possible to shift character borders.

Annotation is done by typing the appropriate letter on the keyboard. After pressing a key a next character candidate is selected and it is thus possible to write fluently. Some special characters has dedicated buttons that simplify the typing. There are buttons for the above mentioned hard to segment pairs such as *ch*. An extra button is available also for character *short s*. The user interface is shown in Figure 2.3

The output of this tool is a set of annotated line images. We also extract individual character images that are saved to directories. We thus can obtain several examples of each character by processing a relatively small amount of data. We also save the positions of character separators that may be useful for successive tasks.

2.1 Requirements

This program has been developed on a Linux system. However, it is possible to use it on all platforms where Python is installed. The supported version is Python 3.5 and higher. The list of required libraries is listed below.

- PIL
- numpy
- Tk - package tkinter

- Opencv - version 3.2 or higher

2.2 User's Guide

The tool is located inside a directory character_segmenter within the downloaded archive. The main script is *segmenter.py*. The execution command is: *python3 segmenter.py* The program reads a configuration file *csg_config.txt*. There are three entries.

- initDir - directory where text lines to be processed are located
- outputDir - directory where extracted characters are stored
- transcriptionDir - directory where transcribed lines and delimiters are saved

Directory btn_images contains special characters that are entered using buttons in the GUI. It can be customized. The provided buttons are intended for transcribing German texts.

Line image is loaded by pressing *ctrl + o* or from menu *File - Open*. Once an image is loaded it appears in the GUI. Depending on its size only a part of it may be displayed. The displayed part is shifted when the visible part is transcribed. The program starts in *Normal mode* which allows transcribing the text simply by typing letters on the keyboard or choosing special characters provided as buttons in the GUI. An extra button is available also for character *short s* to be able to differentiate it from *long s*. It is possible to customize the special buttons simply by adding a new entry to the directory *btn_imgs*. By pressing a key / button the next character candidate is marked by two green vertical lines. Navigation using left and right arrows or buttons *Previous component* or *Next component* is also possible There are basically three scenarios when the user needs an extra effort.

1. splitted character - one character is marked as two or more separate characters because of gaps in the character shape. Clicking button *Merge components* or pressing *ctrl + m* the current character is merged with the following one.
2. merged characters - two or more characters are marked as one because of ligatures or overlaps. Clicking button *Divide component* or pressing *ctrl + d* the current character is divided to two characters with equal sizes.
3. missing character - a candidate character is missing because of incorrect segmentation or mistakes by character merging. Clicking *New*

component a new component can be added directly after the current one.

Left and right border modes are utilized for fine-tuning the position of the character. Switching to one of these modes it is possible to move the left or right merging pressing arrows. Switching back to normal mode is possible by pressing *Escape* or simply typing the annotation of the character.

Saving the prepared annotation is performed by clicking *Save components* or pressing *ctrl + s*.

Chapter 3

Line Annotator

This tool utilizes a trained CRNN model to predict the transcription of a line image. It is then checked by a human annotator and corrected if needed. It uses a simple GUI as shown in Figure 3.2.

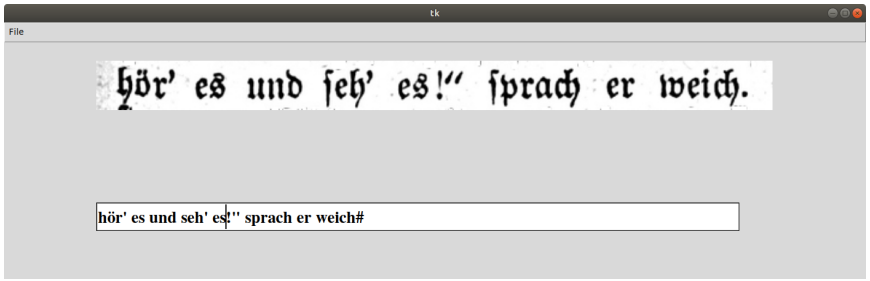

Figure 3.1: Line annotation tool.

The annotation model used was trained on data prepared from the character images extracted by the character segmenter tool. We denote such data as synthetic. The character images extracted from the initial set of lines are used for composition of line images. The texts are taken from old German archives to ensure that the language corresponds with the processed documents. Line images are then composed from the images. We use a simple approach that adds a gap of a fixed size between the characters. A wider gap is used between words. We used 25,000 line images for the initial training. The initial set of real line images obtained also from the character segmenter is used to fine-tune the model. A reasonable accuracy is achieved using only about one hundred of text lines. It is possible to further improve the underlying model by training on newly annotated lines.

3.1 Requirements

This program has been developed on a Linux system. It is possible to use it on all platforms where Python is installed. The supported version is Python 3.5 and higher. The list of required libraries is listed below.

- PIL
- numpy
- Tk - package tkinter
- Pickle
- Itertools
- Opencv - version 3.2 or higher
- Tensorflow
- Keras

3.2 User's Guide

The tool is located inside a directory line_annotator within the downloaded archive. The main script is *annotator.py*. The execution command is: *python3 annotator.py* The program reads a configuration file *la_config.txt*. There are two entries.

- initDir - directory where text lines to be processed are located
- transcriptionDir - directory where transcribed lines are saved

Loading an image is performed by pressing *ctrl + o* or choosing menu *File - Open*. While the image is loading it is also transcribed using stored annotation model. The transcription appears below the line image. It is possible to shift the displayed part of the line image by *ctrl + r* and *ctrl + l*. After the transcribed text is corrected, it is possible to save the transcription pressing *ctrl + s* or using menu *File - Save*.

There are three forms of letter *s* in the German texts we mostly work with. In order to be able to differentiate *short s* and *long s* we use underscore character for the *short s*.

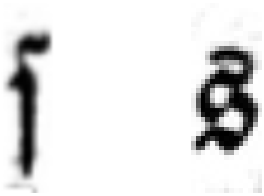

Figure 3.2: Long (left) and short (right) s. Short s is marked as underscore in the transcriptions.

Bibliography

[1] Thomas M Breuel, Adnan Ul-Hasan, Mayce Ali Al-Azawi, and Faisal Shafait, "High-performance ocr for printed english and fraktur using lstm networks," in *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*. IEEE, 2013, pp. 683–687.

[2] Dan Bloomberg, "Leptonica.[online](2010),[cit. 2010-04-25]," 2010.