International Journal on Document Analysis and Recognition (IJDAR)
https://doi.org/10.1007/s10032-019-00347-8

ORIGINAL PAPER

Efficient and effective OCR engine training

Christian Clausner[1] · **Apostolos Antonacopoulos**[1] · **Stefan Pletschacher**[1]

Received: 8 June 2018 / Revised: 16 September 2019 / Accepted: 10 October 2019

Abstract
We present an efficient and effective approach to train OCR engines using the Aletheia document analysis system. All components required for training are seamlessly integrated into Aletheia: training data preparation, the OCR engine's training processes themselves, text recognition, and quantitative evaluation of the trained engine. Such a comprehensive training and evaluation system, guided through a GUI, allows for iterative incremental training to achieve best results. The widely used Tesseract OCR engine is used as a case study to demonstrate the efficiency and effectiveness of the proposed approach. Experimental results are presented validating the training approach with two different historical datasets, representative of recent significant digitisation projects. The impact of different training strategies and training data requirements is presented in detail.

Keywords Optical character recognition · OCR · Machine learning · Training · Graphical user interface · Historical documents

1 Introduction

Document digitisation is an everyday continuing activity at all scales, ranging from the very large content holding institutions (e.g. libraries, archives) to medium-sized operations (e.g. charities, community enterprises) to individuals undertaking small projects. On average, for most use-cases involving relatively simple (no complex backgrounds, no scanning artefacts) modern material, out-of-the-box OCR engines perform very well as they are configured to recognise text written in the most common fonts in the most popular languages.

However, for the multitude of historical documents and for documents written in the many smaller languages in the world, out-of-the-box OCR engines do not perform optimally or even not at all. Some systems allow adjustments via recognition parameters, but this has typically no major impact on results. In such cases, training OCR engines become important in order to recognise those rarer/historic fonts and languages. Even in cases where OCR performs well, training can result in meaningful increases of recognition accuracy—a small percentage of quality increase over a large collection of documents can mean significant savings in manual error correction [1].

Training OCR engines are currently a time-consuming, non-trivial, and disjointed process requiring expert knowledge. With the exception of simply adding a set of unseen symbols to commercially available engines (not real training), the process involves several steps for extensive data preparation, running training scripts, evaluating the performance of the newly trained engine, and repeating the cycle incrementally until sufficient performance improvements are made.

Ideally, an efficient and effective training approach should involve an integrated system where the sequence of data preparation and training steps flows seamlessly and real progress (effectiveness) is evaluated objectively. Through a graphical user interface (GUI), OCR engine training should be a straightforward process, allowing anyone to optimise OCR performance, even for smaller volumes of content in several different circumstances.

This paper describes such a fully integrated OCR engine training approach using the Aletheia document analysis system [2]. All required steps are accessible and controllable through a graphical user interface, including the preparation of training data, OCR engine-specific training and dictionary creation, as well as assessment of the impact of the training by performing OCR and evaluating its results.

✉ Christian Clausner
c.clausner@primaresearch.org

[1] University of Salford, The Crescent, Salford M5 4WT, UK

Published online: 30 October 2019

The description of the proposed system and approach is made using a concrete case study in real-world digitisation scenarios. As such, Tesseract [3] was selected as the example OCR engine of choice, reflecting its popularity with digitisation pipeline developers and users, as well as its open-source ethos and availability to all. It should be noted that for other OCR engines, the most important and resource-intensive steps of training data preparation and evaluation of improvement, as well as the GUI-based system that enables and guides the whole process will remain the same. The only change will be the different OCR engine-specific training scripts that will have to be called from within the Aletheia system, a straightforward addition that can be made by the developers of Aletheia. In fact, an early demonstration, as yet formally unpublished, of the main principles of the system used Gamera [4] as an example OCR engine to be trained.

The efficiency of the proposed system is presented through the description of it and the integrated workflow it enables. The effectiveness of the system is demonstrated through results from experiments on different datasets and application scenarios, including the presence of more than one font in the same document. In addition to the detailed description of the proposed OCR engine training system, this paper also reports on a number of experiments carried out on different datasets to investigate the ideal training conditions in terms of size and quality of a training set.

The next section presents and discusses the background and related work in OCR engine training. Section 3 provides an introduction of the relevant aspects of both the OCR engine to be trained (Tesseract) and the Aletheia system, in which the proposed training approach is incorporated. Each individual aspect of the proposed approach and system is explained in detail in Sect. 4. The effectiveness of the OCR engine training is objectively evaluated, and further experiments are reported in Sect. 5. The paper concludes with a discussion of the experimental results and further remarks in Sect. 6.

2 Background and related work

Optical character recognition systems are typically developed and packaged to support a specific set of languages, scripts, and fonts. Accordingly, if unsupported material is to be processed, adaptations of the recognition engine are required. If the material in question includes new characters and symbols, e.g. in the form of a completely new language and/or script, this training will have to be done from scratch. For less substantial changes, such as adding support for a new font to an already supported language, it might be possible to build on existing resources (language/script models, dictionaries, etc.) within the system to be trained.

Previous large-scale research projects related to mass digitisation [5] have demonstrated the potential gain in accuracy when training OCR engines specifically to the material which is to be processed. Such gains can be achieved even for systems which were designed to follow an Omni-font approach, i.e. not solely relying on comparison of fixed shapes and patterns but employing more flexible features. In [6], for instance, it is reported how recognition rates for non-mainstream documents (Polish historical texts) can be significantly improved by training the OCR engine used. The report states improvements from 45 to 80% character accuracy rate and 15–55% word accuracy rate for typically very challenging to recognise Gothic documents after training ABBYY FineReader on only very few pages.

To leverage OCR training, commercially available systems such as ABBYY FineReader have basic built-in facilities to at least add some new symbols to an existing language/script [7]. It should be noted that this is not training, in the sense of adding completely new scripts, languages and/or dictionaries. Tesseract and other open-source software, on the other hand, offer more training possibilities but still rely largely on scripting of loosely coupled helper tools and/or complex workarounds.

In the literature, there are various reports on how specific OCR training problems were tackled. One example [8] shows how Tesseract was trained to support Ancient Greek. The described process relies mostly on scripting and some manual intervention. The author offers some general recommendations, but neither is a systematic approach related to the choice of parameters and settings followed nor are quantitative justifications given.

Another example can be found in [9] where the authors train Tesseract to recognise Odia, a previously unsupported Indian script. Here, training data is first generated in an artificial way (text entered and typeset in a word processor, then printed and eventually scanned) and then fed into the standard Tesseract command line tools. While this might work for languages (character sets) which are known in principle, it would not be a viable option if unknown characters and symbols are expected to occur (which is quite often the case in historical documents).

Synthetic ground truth [10–13] can be useful for generating large amounts of examples, but it is very hard to guarantee the same level of representativeness and authenticity as datasets containing real-world document images. The biggest downside is that previously unseen/unknown effects can per se never be included in a completely artificial dataset as no discovery stage (like manual ground truthing) is involved. This is a major problem especially for historical documents which are often the subject of (and likely to benefit greatly from) OCR training. Also, realistic rendering of synthetic examples might not be feasible for historical documents if no appropriate fonts are available (font generation can be

an option but requires manual intervention [13]). A viable option could be the combination of manually created ground truth with synthetically derived examples to obtain greater numbers of training data instances.

A more generic approach to training Tesseract based on custom material is presented in [14]. With their main focus being on historical documents, the authors describe a web-based system for generating training data from scanned documents which has the benefit of enabling crowdsourcing in order to allow multiple users to work on the same project. With regard to the actual generation of training data, the system is limited to simple boxes for marking glyphs (rather than allowing more precise polygons), potentially leading to the inclusion of image parts of neighbouring glyphs, which in turn is likely to confuse the training process. The software therefore offers a "brush tool" allowing manual correction of the bitmap images. While manual image microediting this might be a workable solution for a very small amount of samples (or if a very large number of volunteers are available and willing to do it), it appears to be far less efficient than using a polygon-based approach, which could incorporate automatic wrapping of outlines around glyphs and hence producing clean glyph images straight away (e.g. as the Aletheia system [2] does). Also, that system is not fully integrated; requiring users to generate Tesseract OCR profiles by themselves (using scripts and the standard training command line tools) and then uploading them to the so-called Virtual Transcription Laboratory (VTL) portal in order to be used by the provided recognition engine.

Torabi et al. [15] and Christy et al. [16] describe a similarly adaptable approach but use more powerful and efficient tools for glyph image creation. Aletheia [2] is used to prepare those, Franken+ (a system developed by the authors of that paper) is used to create suitable training instances using the glyphs, and finally, Tesseract's command line training procedure is applied. While this enables the training for historical fonts, it is still a very complex operation involving the use of only loosely coupled components, partly with graphical user interface and partly requiring scripting of command line tools. Unfortunately also, the development of Franken+ has not continued and the latest available versions of the other tools are no longer compatible with it. In addition, in terms of the training process there is limited feedback available with respect to the creation of the training set (e.g. with regard to completeness, number of individual character/symbol instances, etc.) making it, inevitably, difficult for the user to keep an overview of training progress and to identify any shortcomings in the data set. It should be noted that the work in described in [15] formed the basis for the inception of the system proposed in this paper, in order to address those issues mentioned above.

The general theme from looking at the existing situation is that there is some support for training OCR engines but typically only in a loosely coupled way making the whole process unnecessarily complicated (for experts and non-technical users of OCR alike) and therefore inefficient. Also, explicit support for training data generation, e.g. in the form of providing statistics, is generally missing thus potentially reducing the quality and effectiveness of training. Finally, several reports provide various recommendations on certain parameters and settings, but it is rare to find empirical data and/or quantitative results with regard to the composition of a good training dataset and the influence of training data selection on the achievable overall results.

3 Tesseract and Aletheia

In the following, the Tesseract OCR engine (including the possibilities of training it) and Aletheia (including its functionality relevant to this work) are outlined. Both complement each other when combined in a general-purpose document analysis and recognition system.

3.1 The Tesseract OCR engine

Tesseract [3] is arguably one of the most popular open-source OCR engine. It features page layout analysis and a flexible text recognition module. It was originally created at Hewlett-Packard between 1985 and 1994 and was open-sourced in 2005. For more detailed information on the capabilities and development history of Tesseract, the reader is referred to [17–20]. Tesseract is still under active development by Google staff and currently available as version 4.0, including a new LSTM-based engine and the traditional engine (selectable via command line option). Both engines have different strengths and weaknesses and therefore are applicable in different use scenarios.

The authors of this paper have chosen the traditional engine (originating from Tesseract 3.0×) as a case study to present the proposed training system as they believe that in several cases (in particular where historical documents are involved) that version will be the engine of choice due to the significant amount of training data required by the LSTM-based engine and the higher resource requirements for running the latter engine.

The more recent versions of Tesseract are fully implemented in the C++ programming language, supporting multiple platforms including Linux- and Windows-based systems. The native releases ship with a command line executable that processes input images and outputs plain text, PDF, or HOCR (an XHTML-based format). Several aspects of the recognition results cannot be stored in those formats and can only be accessed through runtime objects, requiring the usage and extension of the source code.

The official downloadable resource offers pre-trained packages for over 100 languages and variations, but Tesseract can be trained for new languages and character sets. However, as mentioned earlier, the procedure can be complex and requires expert knowledge. Creating training data for historical material is especially challenging as the standard training method is designed to work with fonts that are already available as TrueType or bitmap font files.

The training procedure is described in detail in the following subsection.

3.1.1 Native training procedure

Tesseract can be trained via a collection of command line tools and Linux shell scripts (see [21]). The procedure is composed of two major parts: *shape training* and *dictionary creation* (both explained in more detail below). The output of the training is a "<language>.traineddata" file which needs to be copied to the data folder of the Tesseract instance that will be used to perform OCR taking into account the training data.

In general, the training data generation consists of several steps that produce intermediate files, which, eventually, are packaged to a single results file. The name of that file typically indicates the language it is intended for and possibly font properties—"deu-frak", for instance, is used for German (Deutsch) printed in Fraktur font.

3.1.1.1 Shape training Tesseract's main recognition method uses a feature-based approach to recognise characters. The shape training fulfils two purposes:

- Feature extraction and clustering based on character images (supervised learning).
- Character set creation (i.e. the alphabet).

Several steps are required for shape training—more if no TrueType or OpenType fonts files are available (e.g. in the case of historical documents). Those steps can be outlined as follows:

1. For each font family (includes font variants for bold, italic, etc.) to be added, create an *image—box file pair* (an image with the glyph "shapes" and the corresponding file with the bounding box coordinates and character codes for each of the glyph shapes):
   a. If a TrueType or OpenType font file is available
   i. Call the *text2image* script with a training text sample file and the target font file
   b. Else (no font file available)
      i. Create an image collating all the glyph shapes (complete training set)
      ii. Create a corresponding box file by running Tesseract in box file preparation mode
      iii. Manually correct the box files (for example if a single glyph was wrongly recognised as two characters, the correct character code must be added and the two bounding boxes have to be merged into one and the coordinates corrected)
   c. Run Tesseract in training mode using the image and the corresponding box file (creates a.*tr* file)
2. Generate the Unicode character set (*unicharset*) using all intermediate files from the previous step by running two separate scripts:
   a. Prepare the character set
   b. Add additional properties (e.g. bold, mono-space, etc.)
3. Perform shape clustering (only for Indic languages)
4. Perform feature extraction (called MF training for "Micro Features")
5. Perform character normalisation (called CN training for "Character Normalisation").

3.1.1.2 Dictionary creation Dictionary data in Tesseract is optional but can improve the OCR results significantly (see Sect. 5.5). The character set from the previous step is a requirement to create a dictionary, and therefore, the training has to be performed in this order.

Tesseract uses a special efficient format for dictionaries, called Directed Acyclic Word Graph (DAWG). A DAWG file can be created from a word list in plain text format using the *wordlist2dawg* command line tool. The reverse is also possible (a word list can be extracted from a DAWG file), which can be useful if existing dictionaries are to be updated.

There are different dictionary types which can be created and added individually for each language, to help Tesseract decide the likelihood of different possible character combinations. For instance:

1. Word list: All words that are expected to be encountered (case sensitive).
2. Frequent words: Words that appear often in the given language (e.g. "the" for English).
3. Number patterns: Special patterns representing numbers with mathematical symbols, units, etc.
4. Punctuation patterns: Special patterns representing sentence and word punctuations.
5. Word bigrams: Common combinations of pairs of words.

3.2 The Aletheia document analysis system

Development on Aletheia [2] initially started in 2001 to create a ground truthing system for page layout and text content. Since then, Aletheia has evolved into a full document image analysis system, incorporating support for multiple document file formats, image processing/enhancement and geometric correction methods, several layout description editing modes, integrated OCR, text content entry and manipulation functions, and more. In this section, only those features which are relevant for training Tesseract are described.

Figure 1 shows the general architecture of Aletheia. It has a modular design which allows the integration of different OCR engines. Tesseract is available by default and is part of the software download. The communication between modules is based on a command line interface, and data is interchanged via PAGE XML [22] (other formats such as ALTO and FineReader XML are supported as well).

The core system, including the graphical user interface, is loosely coupled with OCR engines (Tesseract 3 and 4 included by default). The communication is handled as command line calls, and the data interchange is handled via files (temporary files in some cases). The training process is managed and triggered by the core system, but is performed by the OCR engine. The PRImA Tesseract tool provides a command line interface for all the required functions and converts the OCR output to PAGE XML format. The tool was compiled from the Tesseract source code with additions by the authors (PAGE support, etc.). Training is currently only supported for Tesseract 3.

All tools are available for download for academic/personal use [23, 24].

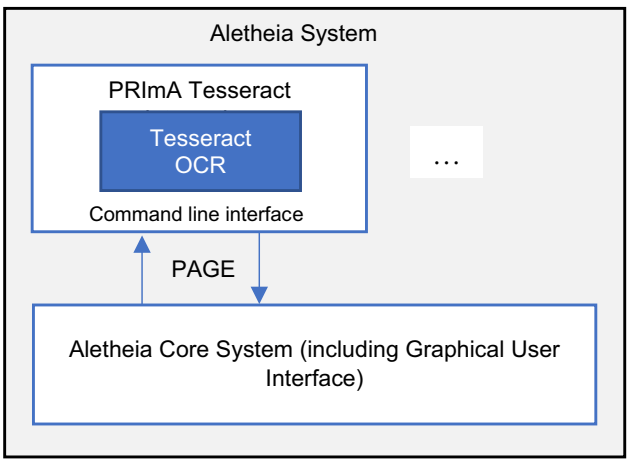


Fig. 1 The architecture of Aletheia

3.2.1 Image operations

Aletheia supports a number of image types and formats; however, Tesseract only accepts bitonal (black-and-white) TIFF files for the training process. Colour or greyscale images can be binarised with one of three methods currently integrated in Aletheia (depending on complexity): Otsu global thresholding [25], Sauvola locally adaptive binarisation [26], or manual thresholding. To improve the creation of training data and the training itself, several image processing methods are available within Aletheia:

- Cropping and rotating
- Automated and manual noise removal
- Manual drawing on the image and erasing
- Exporting to TIFF format.

When working on a document (e.g. to prepare bitonal training data for Tesseract), both the original colour (or greyscale) and a corresponding bitonal image are available for viewing. By being able to quickly switch between the two instances (toggle via a key press), flaws in the bitonal images (e.g. binarisation issues) can be uncovered and, using the image editing tools provided, repaired.

3.2.2 Ground truth production

The Tesseract training process requires ground truth on *glyph* level. In Aletheia, a glyph is a page layout object defined by its shape and position on the page (a polygon) and its associated character code. Glyphs are the lowest level of a layout object hierarchy which starts with *region* objects at the top and progresses through *text line* objects to *word* objects and then glyphs. Here, only text regions are of interest, although other region types such as tables, images, graphics, and separators are available.

It should be noted that sub-glyph components (graphemes) can also be ground truthed with Aletheia and represented in PAGE XML. A use case is presented by Biggs [27].

Producing a sufficient amount of glyph objects (several hundred) can be very significantly labour-intensive when defining and describing each glyph individually. Aletheia supports several strategies and corresponding functionality to reduce the required manual effort. Three possible main strategies are described below, each involving automation assistance in different proportions. Different strategies (or combinations) are applicable in different circumstances depending on the quality of results obtained by the automated methods involved.

3.2.2.1 Segmentation-assisted strategy A user-driven top-down approach can be an efficient way to produce glyph-

level ground truth. Top-down means in this case segmenting the page image into regions, text lines, words, and glyphs in that order. Aletheia offers several automated tools for region, line, and word segmentation (the latter two are an outcome of the IMPACT project [5, 28]). In addition, semi-automated tools are available including precise outline tracing and intelligent (text component preserving) region splitting. If none of the assistive tools work well, for instance, because the image quality is extremely low, all layout objects can also be created and manipulated completely manually (by drawing shapes—simple or complex polygons—around objects and further editing those polygons).

The text content can be provided in a similar way. Entering text at the highest level (regions) is more efficient than entering isolated characters for each glyph individually. When dividing a higher-level object into objects of a level below, a propagation feature allows the corresponding splitting and copying of text from the highest to the lowest level (glyphs) in an automated fashion. In case of discrepancies between the entered text and the corresponding layout objects (such as a different number of words in the text and present word regions), the user is warned and a detailed dialogue for finding and resolving of conflicts is provided.

Several validation and visualisation features of Aletheia help to check the correctness of the created ground truth. This includes among many others the correctness of region outlines (no overlapping objects), completeness of entered text (no regions without text), and inclusion of all textual elements within the reading order.

3.2.2.2 OCR-assisted strategy In Aletheia, the entire page layout object hierarchy and the corresponding text content can be pre-produced using an integrated OCR engine and then (manually) corrected in order to arrive at the required accuracy to be used as ground truth. Tesseract can be used for this purpose, even if itself is to be subsequently trained. Using this strategy, the ground truth production process only involves spotting and correcting segmentation and text recognition errors.

Aletheia includes a wide range of correction and editing tools that are useful in this scenario:

- Resizing and editing of layout object polygons
- Merging and splitting of layout objects
- Text overlay on the image for OCR error checking
- Validation to find polygon overlaps and other problems.

One limitation of this strategy is that Tesseract only produces rectangular glyph objects. This can lead to problems with fonts where the bounding boxes of adjacent glyphs tend to overlap each other. The outlines of the glyphs themselves need to be described precisely (not involving parts of other glyphs) in order to achieve clean training data. Overlapping rectangles would need to be corrected manually (using the tools in Aletheia mentioned above).

3.2.2.3 Hybrid strategy The previous two strategies can be combined in any conceivable way. Layout analysis and OCR can be performed on all levels for specific layout objects. The user could segment a page image into regions manually, for instance, and use Tesseract for all lower level segmentation and text recognition tasks. A recommended approach is to try the automated methods first, and, if unsuccessful, use the undo function of Aletheia and continue using the assisted or manual features for the layout object at hand.

4 The training process with Aletheia

The steps and the user interface for OCR training presented in this paper are tailored to Tesseract. However, the overall workflow is generic with regard to ground truth production, shape and dictionary training, and evaluation. Many aspects of the user interface can be reused directly (ground truthing, selection of shape instances, dictionary editor) or would require small additions (training dialogue). All internal communication from Aletheia to the OCR engine is done via command line calls and XML files already, and new OCR engines can be added with little development effort (dedicated dialogues via GUI editor; training/OCR via command line calls from controller). Since Aletheia supports multiple levels of page elements (regions, text lines, words, and glyphs), training based on whole words or text lines can be implemented as well.

The general training workflow can be outlined as: ground truthing (segmentation and transcription), training input data generation (creating labelled training instances—e.g. character images with Unicode label), shape training (performed by the OCR engine), dictionary creation/tweaking, and (re-) integration of training results (recognition model for OCR engine).

The Aletheia user interface for OCR engine training (Tesseract in this instance) is *pragmatic* and closely tied to the native training process (see Sect. 3.1.1). Figure 2 shows the main dialogue which functions as a dashboard and control panel, providing the following features:

- Naming the training data, typically as <language> [-<special font>] (Fig. 2a)
- Choosing a workspace for the training session, where all intermediate files are stored (Fig. 2b)
- Overview of created/existing training data components (tick boxes indicate whether an intermediate file exists) and functionality for editing where applicable (Fig. 2c)
- Shape training workflow (Fig. 2d; see Sect. 4.3)

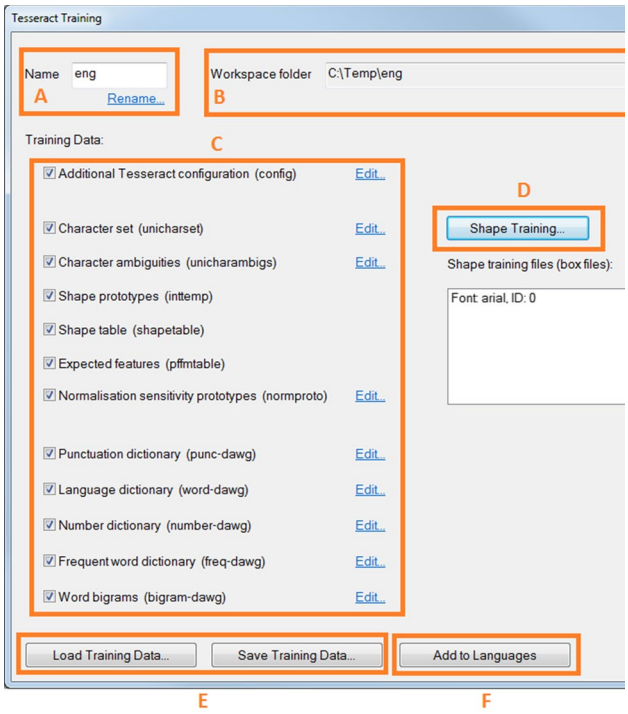


Fig. 2 Main dialogue for Tesseract training in Aletheia

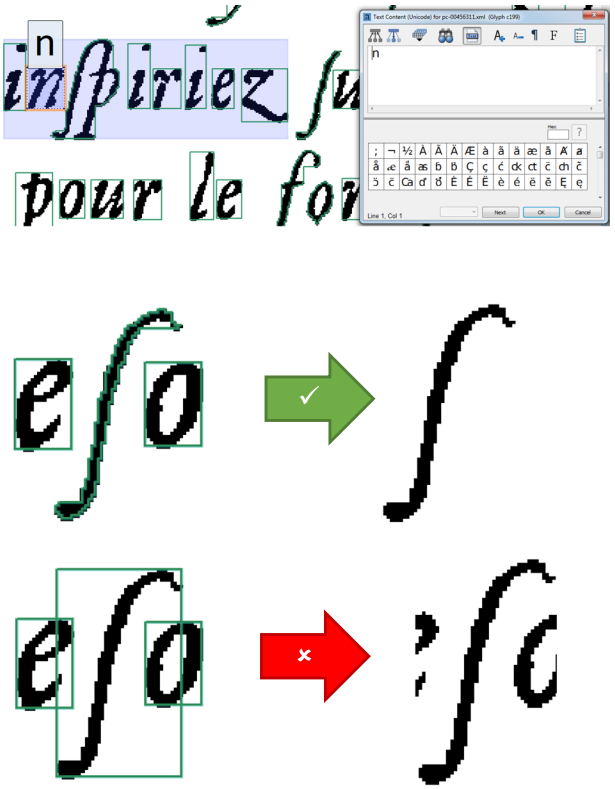


Fig. 3 Glyph description in Aletheia: accuracy of closely fitting polygons versus bounding boxes

- Import from existing training data and export to a training data file (Fig. 2e)
- Creation of training data and adding to Aletheia's integrated Tesseract instance (Fig. 2f)

4.1 Training data selection

The training process in Aletheia targets the more challenging use scenarios, where no font files are available (see Sect. 3.1.1). Instead, training data has to be selected from document images (scans of book pages, for instance). According to the Tesseract training documentation [21], it is recommended to provide 5–30 instances (i.e. glyphs) for each character of the alphabet. This should be used as a main guideline for the selection of images. In addition, all different styles need to be handled separately, so that sufficient character samples are provided for each font and font property (bold, italic, etc.).

Samples of different font sizes can be mixed as long as they are above 10 point. Smaller fonts should be trained individually [21].

It is not straightforward to select suitable training data (for efficient and effective training) purely by visual inspection. To ensure representativeness, the proposed training process provides assistive features (presented in the next two subsections). Those features support an iterative selection process which highlights the need for and possibility for extending and refining the training data set further until the required training conditions are fulfilled.

4.2 Ground truth production

In this step, glyph-level data is produced, using one of the strategies provided by Aletheia, as described in Sect. 3.2.2. The production outcomes are glyphs with a polygonal outline and their corresponding character code as label. Ligatures should be kept as single glyphs and should not be split into their separate constituents. Unicode entries exist for most common ligatures and Aletheia provides a virtual keyboard with a special font (Aletheia Sans) allowing to edit and display ligatures. Similarly, it is possible to work with other uncommon special and/or historical characters which would typically not be supported in standard word processing software (see Fig. 3). In addition to the built-in special font, it is also possible to use other fonts with the virtual keyboard (so long as they are installed/registered within the operating system).

As mentioned earlier, it is important to capture each glyph image individually, avoiding overlaps with other glyphs. Figure 3 (bottom) also shows an example of the extraction result of a glyph with closely fitting polygon in comparison with using a bounding box. The occurrence of adjacent glyphs in

the training data (caused by boxes overlapping more than one glyph) will decrease the recognition performance. In order to avoid this, more precise region outlines in the form of closely fitting polygons can be used.

To facilitate the selection of an appropriately representative training set (as mentioned in Sect. 4.1), Aletheia provides a statistics dialogue which, among other data, lists all characters found within the current document (i.e. page) together with the number of glyph instances. This can be used to make sure all characters of the alphabet are adequately represented.

4.3 Shape training

Having prepared the training data, shape training is the central step in the overall process (for Tesseract, and other OCR engines, which may use different names for the same process). Figure 4 shows the dedicated dialogue in Aletheia. It provides a list of all fonts for which training data has been produced and allows to add, remove, and modify the properties of such data.

Figure 5 shows the training data generation step for a specific font. The font name is arbitrary but has to be unique within the overall training for the current language. Here, the user selects the ground truth files that were created earlier and the corresponding image files. A preview of the extracted glyphs is provided as well as an overview of all characters and corresponding number of instances (glyphs). Prior to starting the shape training step, this overview should be used to check if all relevant characters are represented (letters of the alphabet, punctuation marks, digits, etc.). If not, more training ground truth has to be produced before continuing with the shape training. This additional ground truth can be partial, only covering the missing characters.

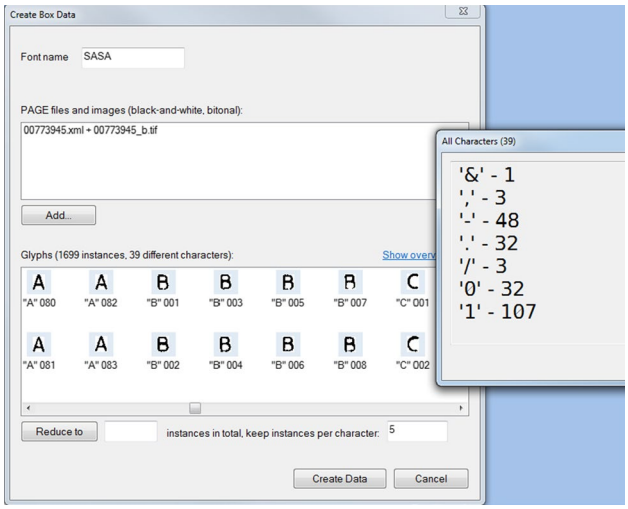


Fig. 5 Shape training step for a selected font

For ease of use, Tesseract's training component is integrated and can be directly executed from the graphical user interface dialogue provided within Aletheia. Once the shape training is complete, the main training dialogue is updated to indicate the existence of the respective intermediate files that have been created.

4.4 Dictionary creation and update

All five types of Tesseract dictionaries can be edited in Aletheia. The editor (see Fig. 6) provides a text field where entries can be directly manipulated and options to filter all words as well as import words from ground truth files or plain text files. The filter options include:

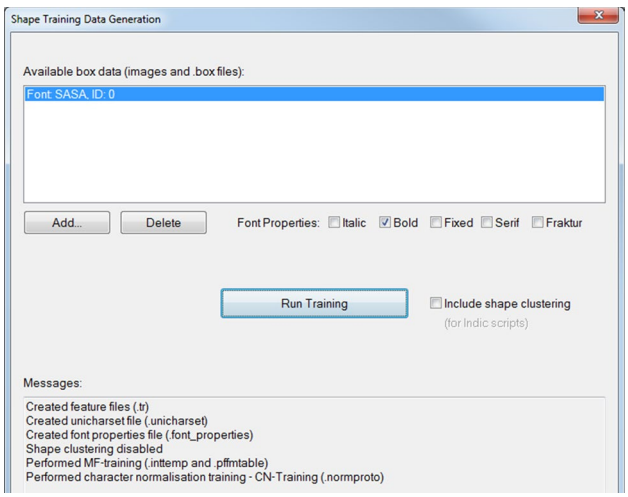


Fig. 4 Shape training dialogue

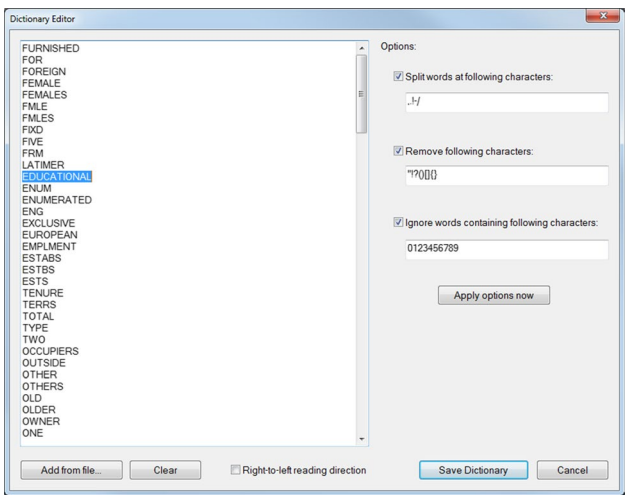


Fig. 6 Dictionary editor

- Splitting of text chunks into words at specified characters
- Removal of specific characters (such as quotation marks)
- Removal of words containing specific characters (such as digits).

The conversion to and from Tesseract's graph format (DAWG) is handled implicitly. If the character sets are compatible, existing dictionaries can be reused. This is useful in cases where only an additional font is to be trained on and not a whole new language.

4.5 Packaging and evaluating the training output

Once training is complete, all resulting information related to a given language (configuration, font, dictionaries, etc., files) can be packaged and exported to a ".traineddata" file which is read in by Tesseract when the respective language is specified (the filename reflects the language name in the official Tesseract executable). The information can also be added directly to the list of available languages of the Tesseract instance that is integrated in Aletheia. A new entry will appear in the page analysis and OCR dialogue (Fig. 7). A more meaningful display name can also be specified.

The new training information can then be evaluated within Aletheia (see Fig. 8) by running and assessing the results of the trained OCR engine. Errors can indicate either missing characters in the training ground truth or unsuitable glyphs (broken or misshaped, for example).

Aletheia offers an OCR checking and correction mode enabling recognised text to be displayed as overlay on top of the scanned document image. For a more efficient comparison, the overlay component approximates the original

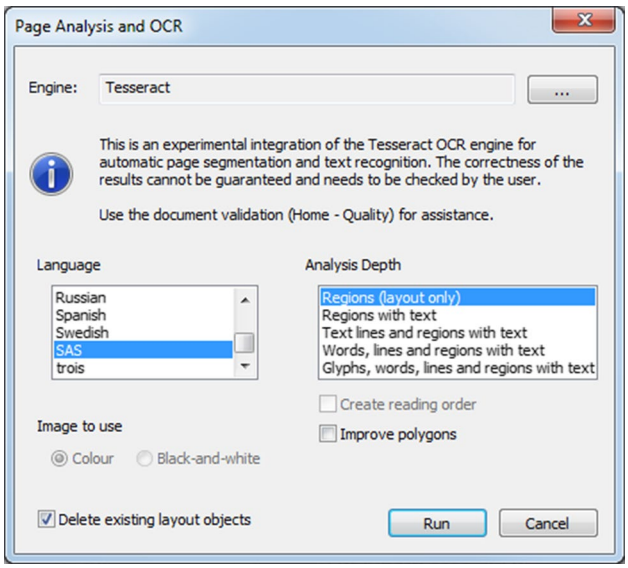


Fig. 7 Page analysis dialogue

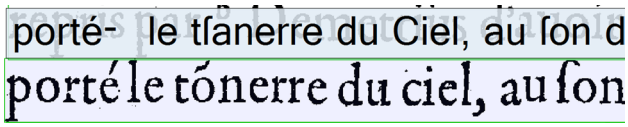


Fig. 8 Test of newly trained OCR engine by visual comparison of its results (top line) to the original text

font size and links the slightly transparent text to the mouse pointer which allows to position it exactly on top of the background image (or wherever it suits the user best). In large-scale digitisation projects (like IMPACT [5] and ENP [29]), it was found that line-by-line comparison, with the recognised text positioned just above the corresponding image text, works well for most users—requiring only little eye movement and avoiding unnecessary distraction.

5 Experiments and Evaluation

The effectiveness of the training process was investigated and validated using two very different datasets, each representing a realistic use scenario where OCR engine training can make a difference: a sample of the 1961 Census for England and Wales (line-printer output resembling more "modern" typewritten/monospaced fonts also), and a historical book from the Bibliothèque National de France (French National Library) dated 1603; collected and ground truthed for the IMPACT project [28], representing an example of typical historical fonts. Figure 9 shows two examples, and Table 1 lists the characteristics and sizes of the training and test sets. The data is available for download [30].

Five different experiments were carried out and reported below, after a brief description of the experimental procedure and the evaluation method used. First, in Sect. 5.3, the results from the trained OCR engine are compared to those of the original untrained engine in order to assess the impact of the training. The second experiment, reported in Sect. 5.4, evaluates the impact of reducing the size of the training dataset (using different strategies) on the quality of training. The improvement in recognition results after the addition of combinations of different types of dictionaries is measured in the third experiment in Sect. 5.5. Finally, the fourth experiment, in Sect. 5.6, examines the impact of training on the recognition of individual fonts within the same document.

5.1 Experimental procedure

All experiments were carried out by creating an initial set of training instances (glyphs with associated character codes), and the results were evaluated using a metric based on text edit distance (see next section).

The ground truth text was transcribed using Aletheia, as described earlier. Special characters and ligatures were entered as seen on the page (diplomatic transcription) and not transliterated.

For the second experiment in particular, several strategies to incrementally reduce the size of the training dataset were applied. More specifically, the following steps were carried out:

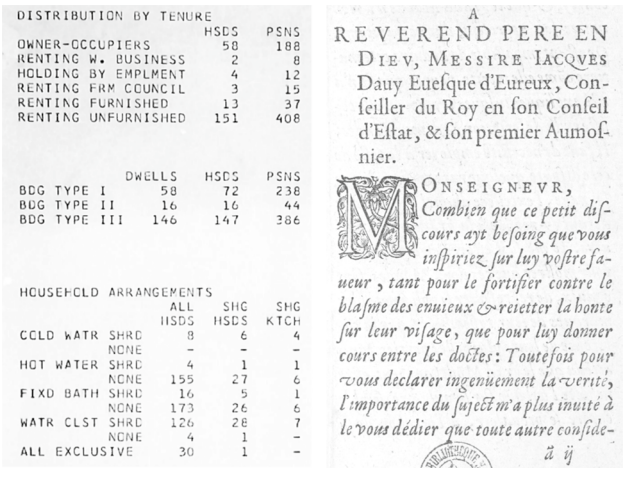
- Page segmentation into text regions, text lines, words, and glyphs consequently
- Text transcription on text region level
- Text propagation from text region level down to glyph level
- (Selection of training instances)
- Shape training
- (Dictionary creation)
- Training data export
- OCR on test set
- Evaluation of result (using ground truth data as reference).

The considered strategies for selecting training instances involved: removal of broken/misshaped glyphs (see Fig. 10), removal of similar looking glyphs, and removal of random glyphs as a control group. The impact of each strategy was tested by gradually removing more and more instances.

5.2 Evaluation methodology

To evaluate the OCR performance on its own, independently from segmentation (layout analysis), text line snippets were favoured over complete pages as input for Tesseract. The respective result texts were recombined to the full-size test set using the original text line order.

To measure the quality of OCR results, a tool called *TextEval* [31] was used (developed by the authors). Among other measures, it uses an implementation of the University of Nevada measure [32] which is based on string edit distance. The evaluation quality reported is a percentage where 100% denotes the text was recognised perfectly. In addition, the TextEval tool provides more in-depth statistical information on which elements of the text were misrecognised.


DISTRIBUTION BY TENURE

	HSDS	PSNS
OWNER-OCCUPIERS	58	188
RENTING W. BUSINESS	2	8
HOLDING BY EMPLMENT	4	12
RENTING FRM COUNCIL	3	15
RENTING FURNISHED	13	37
RENTING UNFURNISHED	151	408

	DWELLS	HSDS	PSNS
BDG TYPE I	58	72	238
BDG TYPE II	16	16	44
BDG TYPE III	146	147	386

HOUSEHOLD ARRANGEMENTS

	ALL HSDS	SHG HSDS	SHG KTCH
CCLD WATR SHRD	8	6	4
NONE	-	-	-
HOT WATER SHRD	4	1	1
NONE	155	27	6
FIXD BATH SHRD	16	5	1
NONE	173	26	6
WATR CLST SHRD	126	28	7
NONE	4	1	-
ALL EXCLUSIVE	30	1	-

A
REVEREND PERE EN
DIEV, MESSIRE IACQVES
Dauy Eueſque d'Eureux, Conſeiller du Roy en ſon Conſeil d'Eſtat, & ſon premier Aumoſnier.

MONSEIGNEVR, *Combien que ce petit diſcours ayt beſoing que vous inſpiriez ſur luy voſtre faueur, tant pour le fortifier contre le blaſme des enuieux & reietter la honte ſur leur viſage, que pour luy donner cours entre les doctes: Toutefois pour vous declarer ingenüement la verité, l'importance du ſuject m'a plus inuité à le vous dédier que toute autre conſide-*

ä ij

Fig. 9 Example pages of the evaluation datasets

5.3 Comparison to default (untrained) OCR

To measure the impact of the training, OCR results with and without training were compared. Without training, in this context, means using the default language information files provided by Tesseract. For completeness of comparison, also the state-of-the-art commercial system ABBYY FineReader Engine 11 was evaluated.

It should be noted that the Census dataset is special in the sense that it only contains upper case Latin characters (as well as digits and punctuation marks). For that reason, an additional performance value is reported, reflecting the use of Tesseract's configuration file where a character white list (allowed characters, a subset of the supported characters) can be specified. While in most everyday cases this feature may not be used, in the case of this particular dataset it should be part of a fair comparison.

Table 2 shows the evaluation results. The trained Tesseract clearly outperforms all other configurations. For both test sets, there is a performance increase of about 5% over the "out-of-the-box" Tesseract OCR results.

5.4 Impact of glyph selection

To study the behaviour of Tesseract when trained with different amounts of training instances, the sizes of the initial training datasets of glyphs were reduced using different strategies.

Table 3 and Fig. 11 show the evaluation results for the Census test set for three different removal strategies to reduce the training dataset:

- Progressive removal of broken and misshaped glyphs (100 at a time, bad cases removed first, see Fig. 10. A manual intervention and therefore subjective with regard to what would be judged broken/low quality glyph images. Criteria were: obviously broken (disjoint) "strokes", thickness variations, deformations, etc.).
- Progressive removal of random glyphs (100 removed each time from the number of glyphs in the previous test).

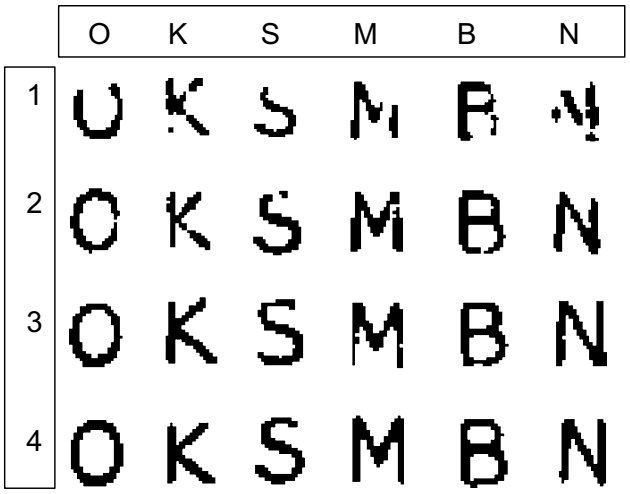


Fig. 10 Broken or misshaped characters (glyphs) progressively selected for removal from the training sets

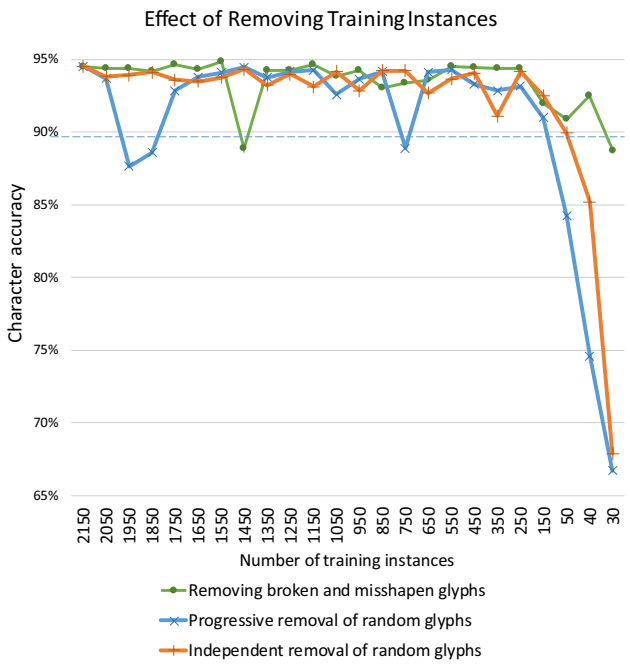


Fig. 11 OCR accuracy for the Census test set after removing training instances (performance of default Tesseract represented by horizontal dashed line)

- Independent removal of random glyphs (starting in each case from the initial set of 2150 glyphs and just removing the required number of glyphs to arrive at the number of training glyphs specified in each test).

Since the removal of training instances was very gradual, the overall number of measurements was deemed sufficient (individual run was not repeated multiple times). Especially, the independent removal of random glyphs represents multiple statistically independent repeats of the experiment. The other two removal strategies reflect how a user would progress the training.

The results show that the training is stable overall, but some outliers can be observed. To find out what is causing the drop of performance for those outliers, the statistics feature of the TextEval tool was used. The result is a table where each column reports: ground truth text snippet, OCR result text snippet, recognition error type (misclassification, miss, false detection) and count. The statistics table revealed that the main reason for the outlier in the "Remove broken glyphs" experiment is the misclassification of all "1"s as "I"s (upper case i). Interestingly, the results improved again when removing more glyphs.

The exact origin of other performance fluctuations is not known because Tesseract can be considered a black box with regard to these experiments. After a careful examination, the source code and documentation might provide further insights.

Figure 12 shows the results for the BNF historical test set using the removal strategy with independent random removals (the more stable strategy based on the Census dataset results). There is more variation in comparison with the Census set, but the trained Tesseract outperforms the default Tesseract in all instances.

A slightly different random glyph removal experiment was carried out with the BNF dataset to investigate if a considerably smaller number of training instances (5–30 per character class as advised by the Tesseract documentation) is indeed of advantage. The experiment was designed to randomly select progressively smaller sets of training instances per character (starting with 90 and ending with 1) from the pool of available glyphs (starting point was 2094 glyphs with

Table 1 Evaluation data sets

Dataset	Characters/font	Training set	Test set
Census	40 Character classes, upper case Latin, digits, and punctuations; one font	2150 glyphs	2115 glyphs
BNF (French National Library)	74 Character classes; Latin with French characters and ligatures, two fonts	773 + 1321 glyphs	2040 glyphs

Table 2 OCR accuracy with and without training

	OCR accuracy by dataset	
	Census (English) (%)	BNF (French) (%)
Default FineReader	88.40	78.09
Default Tesseract	90.08	84.93
Default Tesseract upper case	92.77	N/A
Trained Tesseract	95.40	90.14

Table 3 OCR accuracy for the Census test set after removing training instances (outliers highlighted in bold italic)

Number of training instances (Glyphs)	OCR accuracy by removal strategy		
	Broken removed (%)	Progressively (randomly) removed (%)	Independently (randomly) removed (%)
2150	94.51	94.51	94.51
2050	94.37	93.69	93.83
1950	94.37	***87.65***	93.90
1850	94.17	***88.57***	94.10
1750	94.61	92.84	93.59
1650	94.30	93.79	93.49
1550	94.82	94.07	93.76
1450	***88.85***	94.44	94.30
1350	94.24	93.76	93.21
1250	94.24	94.13	93.98
1150	94.61	94.24	93.11
1050	93.83	92.57	94.13
950	94.20	93.66	92.84
850	93.04	94.13	94.20
750	93.35	***88.88***	94.20
650	93.59	94.10	92.67
550	94.48	94.27	93.66
450	94.44	93.28	94.03
350	94.34	92.84	***91.10***
250	94.37	93.15	94.13
150	91.95	91.00	92.50
50	90.89	84.24	89.91
40	92.50	74.59	85.20
30	88.75	66.71	67.91

varying number of instances per character class). The experiment was repeated five times to reduce statistical variations.

Table 4 shows all results of the experiment. Figure 13 shows the average over all five repetitions and grouped in ranges of two per character instance values resulting in an average value over 10 independent random experiments. As expected, the OCR performance decreases if very few training instances are used. But, for this dataset at least, there is

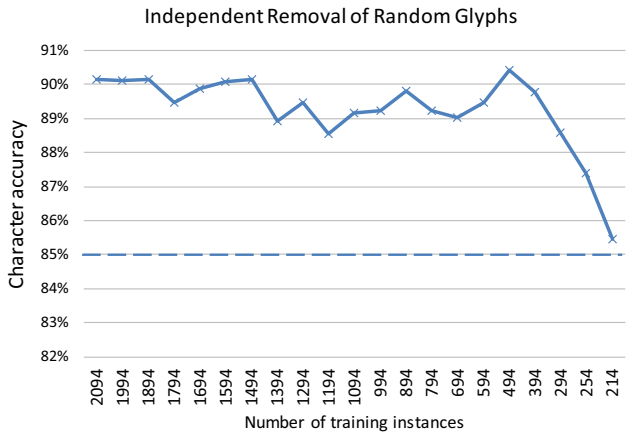


Fig. 12 OCR accuracy for BNF set when removing random training instances (performance of default Tesseract represented by horizontal line)

no statistical evidence that confirms the "less is more" directive from the official documentation. Nevertheless, the training is surprisingly stable, even with significantly less training instances than the initial set. A cap of 10 instances per character, for example, leads to OCR performances between 87.45 and 89.41% (for the five repetitions) as compared to the 90.14% starting point (maximum number of instances).

5.5 Dictionary training results

As mentioned in Sect. 3.1.1, Tesseract supports different types of dictionaries which can be used to improve the OCR results. For historical data, dictionaries are not always readily available. This was the case for the BNF set which uses historical French spelling. Nevertheless, a dictionary for frequent words can be easily created. Because the Census data is repetitive, a dictionary could be created using the training set (all table row and columns headers). A number dictionary was relevant only for the Census data since the French book did not contain any numbers.

Table 5 shows the performance evaluation results. There are measurable improvements when using dictionaries. The frequent word dictionary did not have a positive or negative impact for the Census data, however.

5.6 The impact of training in the presence of more than one font

The BNF dataset contains pages with two different fonts. In this subsection, individual results for each font are presented.

Figure 14 shows the results of the trained OCR engine for both fonts. The plain font, which looks more modern, results in considerably higher OCR accuracy in comparison with the italic font which has many flourishes and significant overlap in the vertical direction between consecutive glyphs.

Table 4 OCR accuracy for random experiments with capped training instances

Max inst. per char[a]	Training inst[b]	OCR accuracy in %				
		Repetition 1	Repetition 2	Repetition 3	Repetition 4	Repetition 5
∞	2094	90.14	90.14	90.14	90.14	90.14
90	1933	89.75	90.11	90.41	90.02	89.96
80	1893	90.50	90.56	90.23	89.78	90.44
70	1853	89.72	89.87	89.41	90.02	90.78
60	1790	89.87	90.11	89.99	90.08	89.96
50	1692	89.74	89.99	89.65	89.90	90.23
40	1563	89.93	89.35	90.66	90.32	89.35
30	1373	89.54	89.51	89.65	90.23	88.99
20	1119	90.08	88.99	90.35	89.75	88.81
10	762	88.69	88.36	87.45	89.41	88.18
1	140	80.82	74.32	79.44	86.03	84.17

[a]Maximum instances per character

[b]Total number of training instances (glyphs)

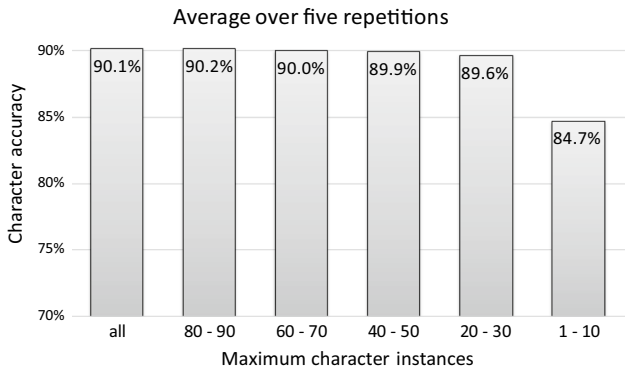


Fig. 13 OCR accuracy for random experiments with capped training instances (grouped in pairs of two)

In both cases, the trained OCR outperforms the standard OCR significantly. In the case of the italic font, the improvement is highest with about 14%.

To find out if there is a correlation in accuracy between the individual fonts within Tesseract, the results from the random experiments were ordered by the OCR performance for one font and the results of the other font were plotted in a scatter graph (Fig. 15). It can be concluded that there is no statistical evidence for any significant correlation (e.g. a drop in performance for one font, due to training variation, does not necessarily result in a drop in performance for the other font).

5.7 Discussion of experimental results

OCR engine training using the proposed approach (for the widely used Tesseract in this case) is effective in the significant majority of cases and can lead to considerable improvements in recognition performance. Nonetheless, drops in quality can occur occasionally, although no complete breakdown (not significantly below the default system) was observed. It is therefore recommended to incorporate a separate final quality assurance step with performance measurement when the training is part of a digitisation project. The positive message is that small changes to the training data can rectify the OCR results and take full benefit of the potential improvement.

Table 5 OCR accuracy with use of different types of dictionaries

	OCR accuracy by dataset	
	Census (Eng) (%)	BNF (French) (%)
Default Tesseract	90.08	84.93
Default Tesseract upper case	92.77	N/A
Trained without dictionary	94.51	89.78
Trained + word dictionary	95.26	N/A
Trained + word and number dictionaries	95.40	N/A
Trained + word, frequent word, and number dictionaries	95.40	90.14

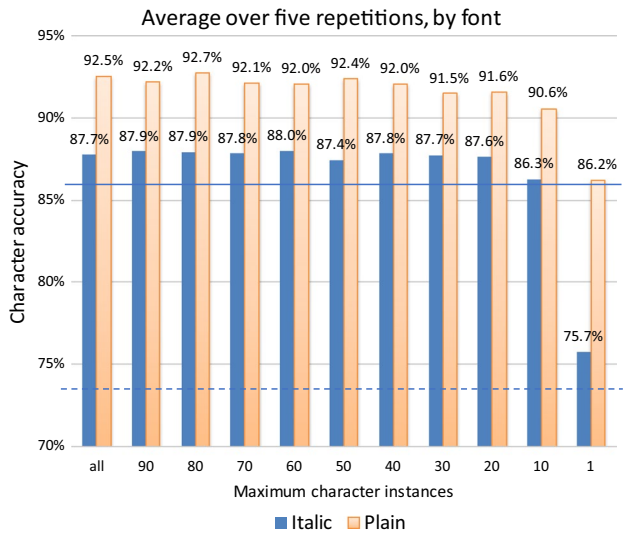


Fig. 14 OCR accuracy by font (italic and plain) for the BNF set (dashed line: default Tesseract, italic; solid line: default Tesseract, plain)

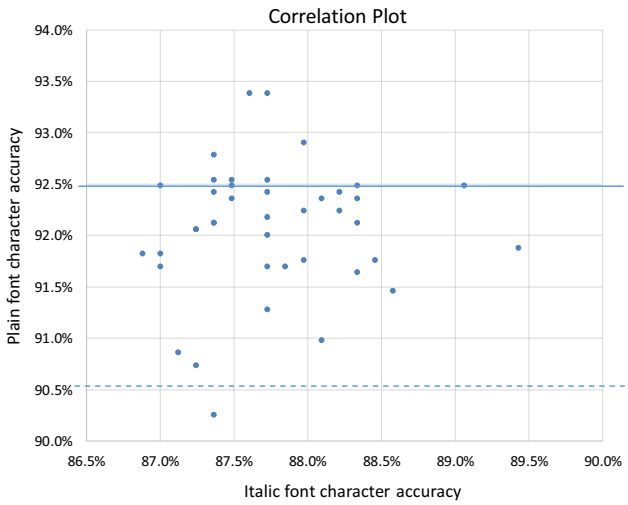


Fig. 15 Correlation plot for the two fonts of the BNF dataset

Throughout the experiments, no universal strategy for selecting suitable training instances could be discovered. The results show that the initial set of training glyphs does not necessarily lead to the best OCR results, but no methodology to remove certain glyphs and retain others can be derived from these experiments (although the independent removal of random glyphs seems more consistent). It can be seen, however, that there is no apparent reason to limit the training instances to between 5 and 30 per character class (as mentioned in the Tesseract documentation). The best approach towards creating a good training dataset still appears to be the proposed iterative refinement process involving sample selection, training, processing, and evaluation.

Given that the presented workflow for training data generation is very efficient (about two seconds per glyph instance), crowdsourcing or generation of synthetic data will not provide significant benefits over the proposed approach, at least for training Tesseract (traditional engine). Crowdsourcing requires a dedicated infrastructure and strict quality assurance measures, resulting in long lead time until usable data becomes available. Synthetic examples can only be produced if adequate fonts can be obtained and they always bear the risk of not being truly representative of the characteristics of the originals at hand. As has been shown, the traditional OCR engine of Tesseract works well with very few training instances per class. This is even more apparent considering the challenges of historical material mentioned in the introduction (no font available, special characters, etc.).

Of course, Aletheia allows to break up the workflow into separate tasks that can be completed by different users. Page segmentation and text transcription can be performed independently and even web-based. (There is an open-source web version of Aletheia [33] that can be deployed.) Intermediate results can be exchanged as PAGE XML. Training and evaluation can be done easily by one user.

6 Concluding remarks

An efficient and effective way to train OCR engines for new languages and fonts was presented. All major components required for the training are integrated into the Aletheia software system, eliminating the need to go through disjointed steps using different scripts, etc. Such a GUI-based approach also allows non-technical users to train OCR engines and benefit their digitisation projects. Moreover, the training data preparation and performance evaluation functionality offered in Aletheia are integral to the proposed iterative training process in order to achieve best results. The training process was shown for the widely used open-source Tesseract OCR (version 4 but using the traditional engine as developed for versions 3.0×) although the process will be largely the same for other engines too: the main data preparation and training evaluation steps are directly applicable in all cases. All mentioned tools are available for download at primaresearch.org [23, 24, 31].

Where necessary (e.g. new material with new character classes or other fonts) the training data can be extended (if the original ground truth is available) and the training can be repeated. While the training data generation is thereby incremental, the actual training is not incremental and performed in full (at least for Tesseract). However, this is not a problem because Tesseract's training method is very fast for the traditional engine. Training for the new LSTM engine is

currently not supported by the presented system, but it can be added in future. This would require adoption of the new training procedures (based on text lines, not glyphs) and modifications to match the changes to the API.

The training approach was evaluated using two different datasets of representative quality (taken from recent large-scale digitisation projects). In both cases, significant improvements over the default Tesseract setup and also over the commercially available FineReader OCR Engine were observed. The experiments reported in this paper also shed a useful light on the intricacies of training in terms of training strategies, the amount of data required and the relative independence of fonts in training.

In practical terms, an experienced user can create training data for a new dataset within a few hours (the overall average time per glyph is 2.1 s for an experienced user, as measured in the experiments). The training script itself runs in only a few seconds. Overall, using training can be recommended, even for small digitisation projects.

References

1. Pletschacher, S., Clausner, C., Antonacopoulos, A.: Europeana newspapers OCR workflow evaluation. In: Proceedings of the 2015 Workshop on Historical Document Imaging and Processing (HIP2015), Nancy, France, pp. 39–46 (2015)
2. Clausner, C., Pletschacher, S., Antonacopoulos, A.: Aletheia—an advanced document layout and text ground-truthing system for production environments. In: Proceedings of the 11th International Conference on Document Analysis and Recognition (ICDAR2011), Beijing, China, pp. 48–52 (2011)
3. Tesseract OCR, https://github.com/tesseract-ocr. Accessed 28 Aug 2019
4. Clausner, C., Pletschacher, S., Antonacopoulos, A.: Efficient OCR training data generation with Aletheia". In: Short Paper Booklet of the 11th International Association for Pattern Recognition (IAPR) Workshop on Document Analysis Systems (DAS2014), Tours, France, pp. 19–20 (2014)
5. IMPACT Project: http://www.impact-project.eu, Accessed 28 Aug 2019
6. Heliński, M., Kmieciak, M., Parkoła, T.: Report on the comparison of Tesseract and ABBYY FineReader OCR engines. PCSS, 2012, oai:lib.psnc.pl:358
7. ABBYY: Recognition with Pattern Training. https://abbyy.technology/en:features:ocr:pattern_training, Accessed 28 Aug 2019
8. White, N.: Training Tesseract for Ancient Greek OCR. Eutypon (28–29), 1–11 (2013)
9. Nayak, M., Nayak, A.K.: Odia characters recognition by training Tesseract OCR engine. In: IJCA Proceedings on International Conference on Distributed Computing and Internet Technology 2014 ICDCIT-2014, (2013)
10. Kanungo, T., Haralick, R.M.: An automatic closed-loop methodology for generating character groundtruth for scanned documents. IEEE Trans. Pattern Anal. Mach. Intell. **21**(2), 179–183 (1999)
11. Baird, H.S.: The state of the art of document image degradation modeling. In: Proceedings of 4th IAPR International Workshop on Document Analysis Systems, Rio de Janeiro, Brazil, pp. 1–16 (2000)
12. Zi, G., Doermann, D.: Document image ground truth generation from electronic text. In: Proceedings of the 17th International Conference on Pattern Recognition, ICPR 2004, vol. 2, pp. 663–666, (2004)
13. Journet, N., Mansencal, B., Visani, M.: Massive, free and reproducible grountruthed document image databases generation with DocCreator. In: Proceedings of 14th IAPR International Conference on Document Analysis and Recognition (ICDAR), pp. 1139–1143, (2017). https://doi.org/10.1109/icdar.2017.188
14. Dudczak, A., Nowak, A., Parkoła, T.: Creation of custom recognition profiles for historical documents. In: Proceedings of the First International Conference on Digital Access to Textual Cultural Heritage (DATeCH'14), ACM, pp. 143–146, http://dx.doi.org/10.1145/2595188.2595209
15. Torabi, K., Durgan, J., Tarpley, B.: Early modern OCR project (eMOP) at Texas A&M University: using Aletheia to Train Tesseract, vol. 23, ACM Press CrossRef. Web, (2013)
16. Christy, M., Gupta, A., Grumbach, E., Mandell, L., Furuta, R., Gutierrez-Osuna, R.: Mass digitization of early modern texts with optical character recognition. J. Comput. Cult. Herit. (2018). https://doi.org/10.1145/3075645
17. Smith, R.: An overview of the tesseract OCR engine. In: IEEE Computer Society Proceedings of the 9th International Conference on Document Analysis and Recognition (ICDAR2007), 2007, pp. 629–633
18. Smith, R., Antonova, D., Lee, D.S.: Adapting the Tesseract open source OCR engine for multilingual OCR. In: Proceedings of the International Workshop on Multilingual OCR, Barcelona, Spain, (2009), https://doi.org/10.1145/1577802.1577804
19. Smith, R.: History of the Tesseract OCR engine: What worked and what Didn't. In: Proceedings SPIE 8658, Document Recognition and Retrieval XX, 865802, https://doi.org/10.1117/12.2010051 (2013)
20. Shams, S.: Breaking down Tesseract OCR. Online article in Machine Learning Medium, https://machinelearningmedium.com/2019/01/15/breaking-down-tesseract-ocr/. Accessed 28 Aug 2019
21. Training Tesseract. https://github.com/tesseract-ocr/tesseract/wiki/Training-Tesseract, Accessed 28 Aug 2019
22. Pletschacher, S., Antonacopoulos, A.: The PAGE (page analysis and ground-truth elements) format framework. In: Proceedings of the 20th International Conference on Pattern Recognition (ICPR2010), Istanbul, Turkey, IEEE-CS Press, pp. 257–260 (2010)
23. Tesseract OCR to PAGE (command line tool): primaresearch.org/tools/TesseractOCRToPAGE, last access 23/07/2019
24. Aletheia (Lite and Pro): primaresearch.org/tools/Aletheia, Last access 23/07/2019
25. Otsu, N.: A threshold selection method from gray level histogram. IEEE SMC-9, No. 1, pp. 62–66, 1979
26. Sauvola, J., Pietikäinen, M.: Adaptive document image binarization. Pattern Recognit. **33**(2), 225–236 (2000)
27. Biggs, J.: Comparison of visual and logical character segmentation in Tesseract OCR Language Data for Indic Writing Scripts. ALTA 11–20 (2015)
28. IMPACT Centre of Competence for Digitisation in Europe. In: Proceedings of the Australasian Language Technology

Association Workshop 2015—ALTA, Parramatta, Australia. http://www.digitisation.eu/. Accessed 28 Aug 2019
29. Europeana Newspapers: http://www.europeana-newspapers.eu. Accessed 25 May 2018
30. Training and test data for Tesseract Training: https://www.primaresearch.org/datasets/TESSERACT_TRAINING, Accessed 23 July 2019
31. PRImA Text Evaluation tool. http://www.primaresearch.org/tools/PerformanceEvaluation. Accessed 25 May 2018
32. Rice, S.V.: Measuring the accuracy of page-reading systems. Ph.D. Thesis, University of Nevada, Las Vegas (1996)
33. WebAletheia: https://github.com/PRImA-Research-Lab/prima-aletheia-web. Accessed 11 Mar 2019