Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/pr

DETEXTIVE optical character recognition with pattern matching on-the-fly

Ursina Caluori, Klaus Simon*

Swiss Federal Laboratories for Materials Testing and Research, EMPA, Überlandstrasse 129, Dübendorf, Switzerland

ARTICLE INFO

Article history:
Received 18 October 2013
Received in revised form
8 May 2014
Accepted 30 August 2014
Available online 10 September 2014

Keywords:
OCR
Pattern matching
Historic prints
Black letter fonts
Mass-digitization
Performance test

ABSTRACT

In this paper we present a new OCR-concept designed for the requirements of historic prints in the context of mass-digitizations. The core part is the glyph recognition, based on pattern matching with patterns that are derived from computer font glyphs and are generated on-the-fly. The classification of a sample is organized as a search process for the most similar glyph pattern. This results in consistently good hit rates for arbitrary fonts without any training. In particular, we investigate the performance of our prototype in comparison to popular commercially available OCR-software.

1. Introduction

Recently, mass-digitization of historic prints has become a popular issue for libraries. For instance, in the *European Digital Library* program the EU strives for renewing Europe's printed heritage as digitally available resources where the final challenge is *transforming digital images of scanned books into electronic text*. However, the available OCR-technology does not cope with this task and therefore the EU launched considerable research activities like IMPACT[1] in order to improve the situation.

Modern software for *optical character recognition* (*OCR*) like *neural networks* or *support vector machines* [3,5,10–12,22] follows the *machine learning* approach. The graphical appearance of a character is allowed to vary in a certain range where its common or typical shape has to be learned in a supervised training process. The main advantage of this software concept is generality, i.e. it can give reasonable answers in many cases and the proceeding is standardized to a large extent. On the other hand, the recognition accuracy depends on the used training set and, therefore, is subject to principal restrictions. Furthermore, the training process itself is demanding.

The old fashioned way of pattern recognition is *pattern matching* where a sample image is compared with a pattern pixel by pixel. If all or almost all pixels of the sample match a specific glyph pattern, then the sample is identified. This algorithmically simple method is highly accurate and works for arbitrary patterns without training or other preparations. One drawback is the involved image processing which overstrained the early computer technology such that pattern matching was replaced by more efficient concepts in the 1970s. Secondly, the patterns—the glyph images of a font character—were usually hard coded. Hence, the software could only recognize the characters of specific fonts, for instance OCR-A (1968) or OCR-B (1972) which were exclusively designed[2] for this purpose, see [13] for survey.

In the meantime, computer performance has improved by a factor of more than a 1000 both in running time and storage capacity which overcomes the first problem. The solution for the second problem arose from the desktop publishing developed in the mid 1980s. A computer font became a PostScript data format which is rendered by a PostScript interpreter (*raster image processor* (*RIP*)). The integration of a RIP into a printer allows it to handle arbitrary fonts. As a consequence, treating fonts as an input parameter became a common standard in publishing, see [7,16].

The improved computer performance in combination with the integration of a RIP into an OCR-software suggests a renewal of pattern matching for mass-digitizations of historic prints. Typical

* Corresponding author.
E-mail addresses: ursina.caluori@empa.ch (U. Caluori), klaus.simon@empa.ch (K. Simon).

[1] Citation from http://www.impact-project.eu: Automated text recognition, carried out by Optical Character Recognition (OCR) engines does in many cases not produce satisfying results for historical documents. Recognition rates are poor or even useless. No commercial or other OCR-engine is able to cope satisfactorily with the wide range of printed materials published between the start of the Gutenberg age in the 15th century and the start of the industrial production of books in the middle of the 19th century.

[2] See ISO 1073-1:1976 or DIN 66008.

http://dx.doi.org/10.1016/j.patcog.2014.08.026
0031-3203/© 2014 Elsevier Ltd. All rights reserved.

for such applications is a constant layout for many thousands of scans showing unusual metal fonts.[3] The stability of the layout allows a professional operator to adjust the OCR-software to a specific task by assigning the involved fonts as an input to the program.

During the last 3 years the authors have developed an OCR-prototype software, called the Detextive, based on pattern matching with on-the-fly generated glyph patterns. Recently, we have shown that the concept is functional, see [17–19]. In this paper we conclude that these conference papers describe some new improved similarity measures and present a performance test with the popular OCR-softwares Abbyy FineReader, Adobe Acrobat and Google Tesseract. The test considers 62 characters from 112 different fonts in several font sizes and scan resolutions for relatively noisy data.

In our test Detextive outperforms the competitors on the vast majority of considered fonts and is at least comparable in each case. In contrast to the others, Detextive keeps a more or less steady high hit rate throughout the entire font set. Obviously, the competitors are optimized for some basic fonts like Helvetica or Times New Roman and neglect the rest. Thereby, Detextive seems to be less susceptible to noise.

In Section 2 we describe our concept in greater detail. Then we consider several improved similarity measures which are of vital importance for our approach. Next, we introduce the realized recognition test for glyphs and analyze our results. Finally, we give an outlook and draw conclusions.

2. Algorithm and data structures

The usual result of a scanning process is a gray image which has to be binarized to 0 (black) for textual parts and 1 (white) for the background. Black pixels are called *dots* and adjacent dots are summarized into *segments*. Accordingly, consecutive dots are gathered in distinct intervals which are organized in linear lists and sorted in relation to their endpoints. In each case, the used coordinates are relative to the corresponding bounding box. Segments are stored in a run length encoding both in horizontal and vertical direction. An example is given in Fig. 1.

One or more[4] segments of the input file belong to a character. Such a group is denoted a *sample* and the main task of an OCR-system is the assignment of a fitting glyph to it, called the *classification* of the sample. A *pattern*—or more exactly, a *glyph pattern*—is formed by the segments found in a glyph image where a glyph is a specific graphical representation of a character. In our context, the considered glyphs are described as vector graphics and can be rendered to a glyph image in the corresponding sample size with a RIP. The associated font, the corresponding character, the used pt-size and the induced bounding box are stored together with the glyph image in each pattern's representation. The set of currently considered fonts and font sizes is assigned by the user.[5]

Popular concepts of classification like support vector machines are based on *features* which are weighting functions applied to samples or other groups of segments. The features form a *feature vector* which represents the corresponding sample in an n-dimensional *feature space* which is separated into distinct regions, called *classes*, in accordance with the characters to recognize. For that reason, *classifying a sample* means finding out the class which contains its feature vector. Obviously, the algorithmic way to do this depends on the representation of the classes.

In machine learning a class is usually determined by one or more representative examples. Let them be called *class instances*.

[3] Often black letter (fraktur) fonts.
[4] Like in an "i".
[5] As usual in desktop publishing.

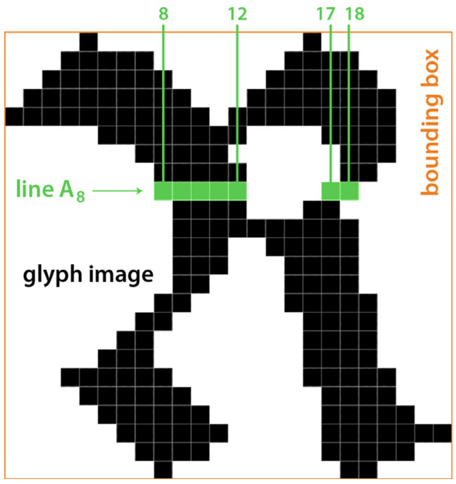

Fig. 1. Run length encoding in horizontal direction. The list A_8 contains the intervals [8,12] and [17,18].

Then a simple classification of a sample is given by the nearest distance between the feature vector of the sample and those of class instances. Consequently, classes are induced by the *voronoi diagrams* of the feature vectors of all class instances, see [2, Chapter 7].

Related to this scenario, there are two principal ways for the classification. The *explicit* approach first computes the voronoi diagrams and the classification is then performed as *point location*, see [2, Chapter 6]. In particular, support vector machines transform additionally the underlying feature space in order to optimize the point location. The first phase, the derivation of the classes from its instances, is well-known as *supervised training* and is part of the software development. This is different from the second phase in which the performing of a classification as some kind of point location means applying the resulting software by the user.

Contrarily, the second approach for the classification of a sample, denoted as *implicit*, calculates simply all distances between the sample and all class instances. The nearest one determines the class of the sample. A main advantage of this concept is that not an explicit class representation is needed and, therefore, the training phase falls upon. Moreover, the set of class instances can be treated as an input parameter of the software and be chosen by the user. Note that the implicit approach requires only an estimable distance function. Hence, the (Euclidean) metric between feature vectors can be replaced easily by other measures between sample and class instances.

Our classification of a sample is an implicit search for the most similar glyph pattern where the glyph patterns correspond to the class instances. At first, we have to describe the comparison between a sample S and a pattern P. For that reason, we calculate the projection profiles of S and P where a *projection profile* is given by the number of pixels per line in horizontal or vertical direction. The best sample-pattern overlapping is determined by the maximum cross correlation between corresponding projection profiles. The horizontal (vertical) projection profile of S is denoted as X_S (Y_S) and X_P (Y_P) analog for P.

In the following, we assume, without loss of generality, that the overlapping between S and P is fixed and the relative coordinates for the row and column lists have been adapted to the common bounding box *CBB*. The typical operation for computing similarity measures between S and P is comparing the pairs of rows (A_i, B_i), $i = 0, \ldots, t-1$, or, respectively, the pairs of columns (C_j, E_j), $j = 0, \ldots, k-1$, where $t \cdot k$ is the size of *CBB*, A_i and C_j (B_i and E_j) belong to S (P).