RUSSIAN-ARMENIAN UNIVERSITY

INSTITUTE OF MATEMATICS AND INFORMATICS

DEPARTMENT OF MATEMATICS AND MATEMATICAL MODELLING

Research and Development of OCR Tools for the Armenian Language

Master's Thesis in Applied Mathematics and Informatics

[illegible]
____
signature

1nd year Master's student:
Karen Nikoghosyan

[illegible]
____

Supervisor:
Tsolak
Ghukasyan

signature

Yerevan, 2019

Contents

Introduction

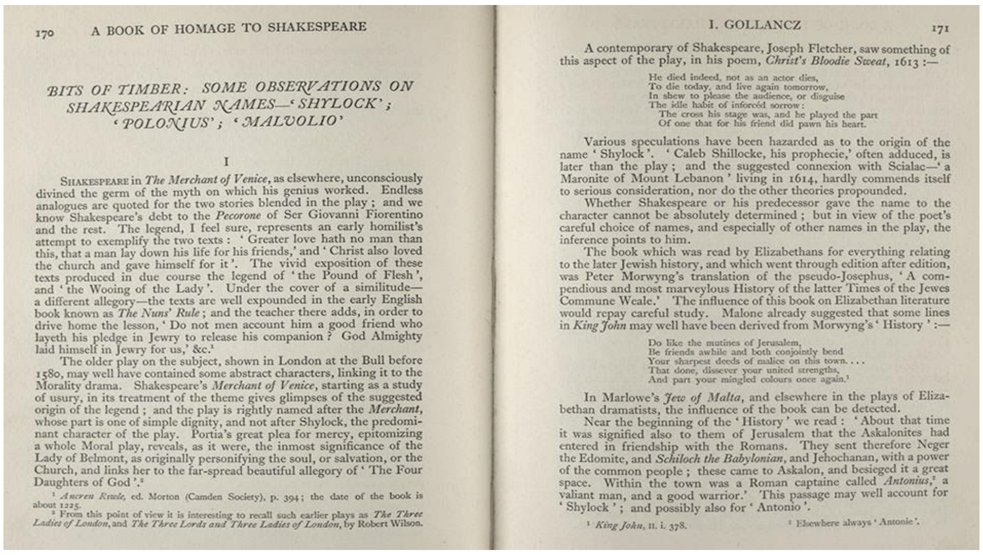
Optical character recognition or OCR[1] refers to a set of computer vision [2] problems that require us to convert images of digital or hand-written text images to machine readable text in a form your computer can process, store and edit as a text file or as a part of a data entry and manipulation software. The images can include documents, invoices, legal forms, ID cards or OCR in the wild like reading street signs, shipping container numbers or vehicle number plates.

OCR can be used in common industries and applications including reading serial numbers in automotive or electronics applications, passport processing, secure document processing (checks, financial documents, bills), postal tracking, publishing, consumer goods packaging (batch codes, lot codes, expiration dates), and clinical applications. Also OCR readers and software can be used, as well as smart cameras and vision systems which have additional capabilities like barcode reading and product inspection.

OCR is still a challenging problem especially when text images are taken in an unconstrained environment. It is about complex backgrounds, noise, lightning, different font, and geometrical distortions in the image.

Challenges in the OCR problem arise mostly due to the attribute of the OCR tasks at hand. Generally OCR tasks divide into two categories:

Structured Text Fig. 1**:** Text in a typed document. In a standard background, proper row, standard font and mostly dense.



170 A BOOK OF HOMAGE TO SHAKESPEARE

BITS OF TIMBER: SOME OBSERVATIONS ON SHAKESPEARIAN NAMES—'SHYLOCK'; 'POLONIUS'; 'MALVOLIO'

I

SHAKESPEARE in *The Merchant of Venice*, as elsewhere, unconsciously divined the germ of the myth on which his genius worked. Endless analogues are quoted for the two stories blended in the play; and we know Shakespeare's debt to the *Pecorone* of Ser Giovanni Fiorentino and the rest. The legend, I feel sure, represents an early homilist's attempt to exemplify the two texts: 'Greater love hath no man than this, that a man lay down his life for his friends,' and 'Christ also loved the church and gave himself for it'. The vivid exposition of these texts produced in due course the legend of 'the Pound of Flesh', and 'the Wooing of the Lady'. Under the cover of a similitude—a different allegory—the texts are well expounded in the early English book known as *The Nuns' Rule*; and the teacher there adds, in order to drive home the lesson, 'Do not men account him a good friend who layeth his pledge in Jewry to release his companion? God Almighty laid himself in Jewry for us,' &c.[1]

The older play on the subject, shown in London at the Bull before 1580, may well have contained some abstract characters, linking it to the Morality drama. Shakespeare's *Merchant of Venice*, starting as a study of usury, in its treatment of the theme gives glimpses of the suggested origin of the legend; and the play is rightly named after the *Merchant*, whose part is one of simple dignity, and not after Shylock, the predominant character of the play. Portia's great plea for mercy, epitomizing a whole Moral play, reveals, as it were, the inmost significance of the Lady of Belmont, as originally personifying the soul, or salvation, or the Church, and links her to the far-spread beautiful allegory of 'The Four Daughters of God'.[2]

[1] *Ancren Riwle*, ed. Morton (Camden Society), p. 394; the date of the book is about 1225.

[2] From this point of view it is interesting to recall such earlier plays as *The Three Ladies of London*, and *The Three Lords and Three Ladies of London*, by Robert Wilson.

I. GOLLANCZ 171

A contemporary of Shakespeare, Joseph Fletcher, saw something of this aspect of the play, in his poem, *Christ's Bloodie Sweat*, 1613:—

He died indeed, not as an actor dies,
To die today, and live again tomorrow,
In shew to please the audience, or disguise
The idle habit of inforcéd sorrow:
The cross his stage was, and he played the part
Of one that for his friend did pawn his heart.

Various speculations have been hazarded as to the origin of the name 'Shylock'. 'Caleb Shillocke, his prophecie,' often adduced, is later than the play; and the suggested connexion with Scialac—'a Maronite of Mount Lebanon' living in 1614, hardly commends itself to serious consideration, nor do the other theories propounded.

Whether Shakespeare or his predecessor gave the name to the character cannot be absolutely determined; but in view of the poet's careful choice of names, and especially of other names in the play, the inference points to him.

The book which was read by Elizabethans for everything relating to the later Jewish history, and which went through edition after edition, was Peter Morwyng's translation of the pseudo-Josephus, 'A compendious and most marveylous History of the latter Times of the Jewes Commune Weale.' The influence of this book on Elizabethan literature would repay careful study. Malone already suggested that some lines in *King John* may well have been derived from Morwyng's 'History':—

Do like the mutines of Jerusalem,
Be friends awhile and both conjointly bend
Your sharpest deeds of malice on this town. . . .
That done, dissever your united strengths,
And part your mingled colours once again.[1]

In Marlowe's *Jew of Malta*, and elsewhere in the plays of Elizabethan dramatists, the influence of the book can be detected.

Near the beginning of the 'History' we read: 'About that time it was signified also to them of Jerusalem that the Askalonites had entered in friendship with the Romans. They sent therefore Neger the Edomite, and *Schiloch the Babylonian*, and Jehochanan, with a power of the common people; these came to Askalon, and besieged it a great space. Within the town was a Roman captaine called *Antonius*,[2] a valiant man, and a good warrior.' This passage may well account for 'Shylock'; and possibly also for 'Antonio'.

[1] *King John*, II. i. 378.

[2] Elsewhere always 'Antonie'.

Fig. 1. Structured Text

Unstructured Text Fig. 2**.** Text at random places in a natural scene. Sparse text, no proper row structure, complex background , at random place in the image and no standard font .

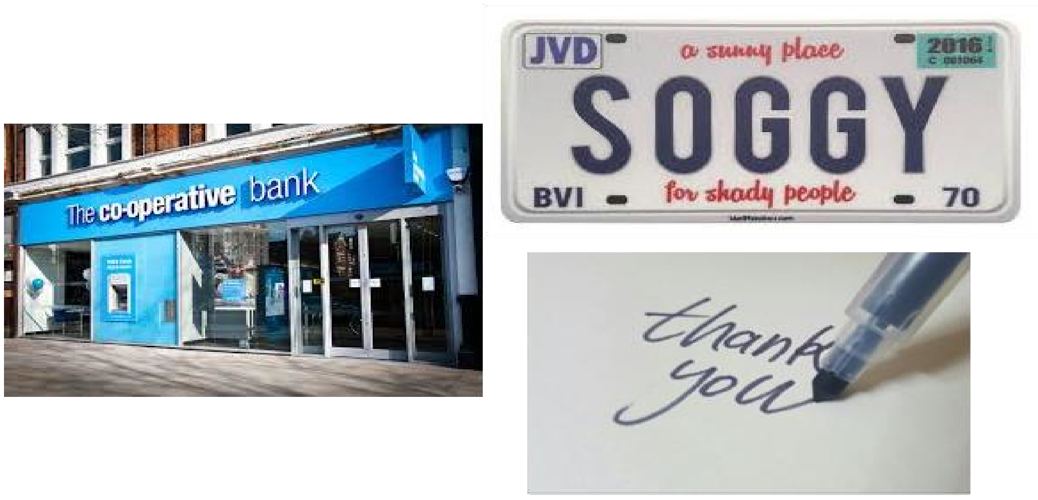


Fig. 2. Unstructured Text

There are many different approaches to solving the optical character recognition problem. One of the most common and popular approaches is based on neural networks, which can be applied to different tasks, such as pattern recognition, time series prediction, function approximation, clustering, etc.

Deep learning approaches have improved over the last few years, reviving an interest in the OCR problem, where neural networks can be used to combine the tasks of localizing text in an image along with understanding what the text is. Using deep convolutional neural architectures and attention mechanisms and recurrent networks have gone a long way in this regard.

Research objectives

The aim of this work is to create an OCR tool for the Armenian language that will allow you to receive high quality texts.

In the first stage of the work it is planned to do research, find OCR tools working for Armenian, study those tools, understand in which cases it works badly for Armenian language and try new solution methods by proposing to create a better OCR tool.

Research has shown that there are various OCR tools that work for the Armenian language to a greater or lesser extent, but not all of them are open. The most famous of them is Tesseract.

Tesseract is an OCR tool that works for many languages, including Armenian.

Studies have shown that the effectiveness of Tesseract declines sharply in the case of rare fonts of low-quality images (fe.g. ‘ւոեււակը’ instead of ‘տեսակը’;

'փոււից' instead of 'փուլից').

Tesseract System Architecture and Data Structures

Introduction

Tesseract[3] is an open source optical character recognition (OCR) platform. OCR extracts text from images and documents without a text layer and outputs the document into a new searchable text file, PDF, or most other popular formats. Tesseract is highly customizable and can operate using most languages, including multilingual documents and vertical text. Tesseract has unicode (UTF-8) support, and can recognize more than 100 languages "out of the box".

Tesseract supports various output formats: plain text, hOCR (HTML), PDF, invisibletext-only PDF, TSV. The master branch also has experimental support for ALTO (XML) output.

Tesseract can be trained to recognize other languages.

Tesseract System Architecture

The pipeline of Tesseract OCR engine is given in Fig. 3. The first step is Adaptive Thresholding, which converts the image into a binary version. The next step is page layout analysis, which is used to extract the text blocks within the document as shown in Fig 5. In the next stage the baselines of each line are detected and the text is divided into words using definite spaces and fuzzy spaces [8]. In the next step, the character outlines are extracted from the words. Recognition of text is then started as a two-pass process. In the first pass,word recognition is done using the static classifier. Each word passed satisfactory is passed to an adaptive classifier as training data [8]. A second pass is run over the page, using the newly learned adaptive classifier in which words that were not recognized well enough are recognized again.

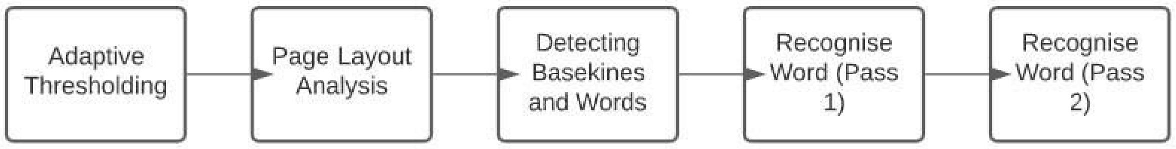


Fig. 3. Architecture of Tesseract OCR

Adaptive Thresholding

In image processing, thresholding is applied to obtain binary images from grayscale images. Adaptive thresholding is found to be better as compared to conventional thresholding technique. In an image, some parts remain under more shadow and often illuminations also affect the image. In conventional thresholding methods, a global or standard threshold value is taken as mean value. In an image, if a darker part or pixels contain a value larger than the threshold value, then that part of the image appears in the foreground. Similarly if the value is less than the threshold value, then that pixel or part appears in the background. Binary image is the resultant image of adaptive thresholding as it depicts the differences between different threshold values.

Tesseract OCR internally applies Otsu binarization method[4]. However this method selects an optimal global threshold according to image histogram. If there is a shadow on the image, the tesseract will fail extracting the characters. Fig. 4.1 and Fig 4.2 are examples of image adaptive thresholding.

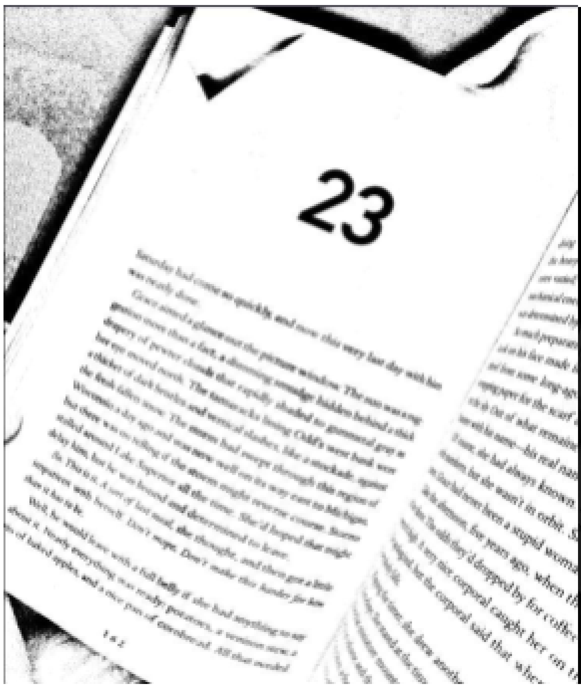
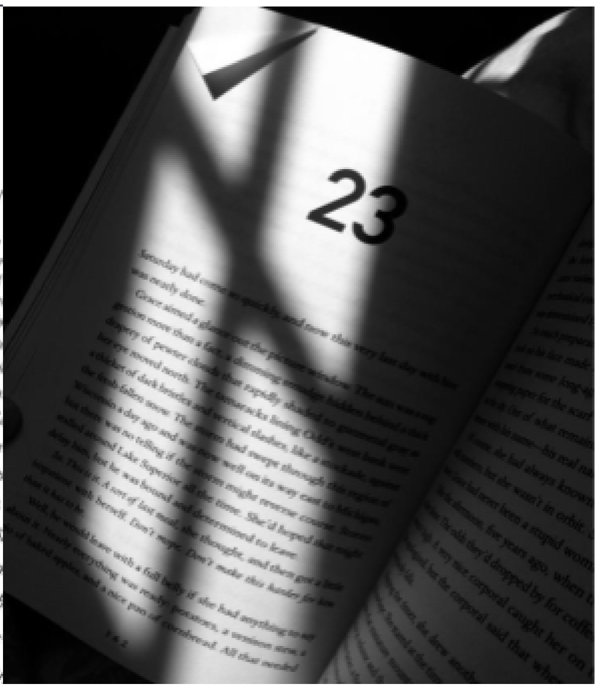



Fig 4.1 Thresholded image Fig 4.2 Original image

Page Layout Analysis

One of the first steps of OCR is the Page Layout Analysis phase. During this, the image is divided into text-containing, non-text-containing parts, just as multi-column text is divided into columns. At Tesseract, this process is based on detecting tab-stops in a document image. It has the following steps.

The first stage is morphological processing, this is done using Leptonica[7]. During this process Leptonica detects the vertical lines and the images. These elements are removed from the input image before passing the image to connected component analysis.

During the second stage, components which are connected with tag-stops are found and then grouped into tab-stop lines. After that scanning all connected components and classified into Column Partitions (CP), subject to the constraint that no CP may cross a tab stop line. Further chains of text CPs are divided into groups of uniform line spacing, which make text blocks. After that each chain of CPs represents a candidate region. The reading order of the regions are determined by some heuristic rules

Here is sample code for on how to use library for page layout analysis on an image:

```
tesseract::TessBaseAPI tessApi;
tessApi.InitForAnalysePage();

tessApi.SetImage(...);

tesseract::PageIterator *iter = tessApi.AnalyseLayout();

// Instead of RIL_WORD you can use any other PageSegMode
while (iter->Next(tesseract::RIL_WORD)) {
  int left, top, right, bottom;

  iter->BoundingBox(
        tesseract::RIL_WORD,
        &left, &top, & right, & bottom
  );
```

First, initialize **TessBaseAPI** instance. You can either use **Init()** (if you want to perform further text recognition) or **InitForAnalysePage()** (if you're interested just in text boxes).Second, set the image using **SetImage().** And finally, call **AnalyseLayout()** to get

PageIterator which provides you with text boxes. Fig. 5 is example of Page Layout Analysis

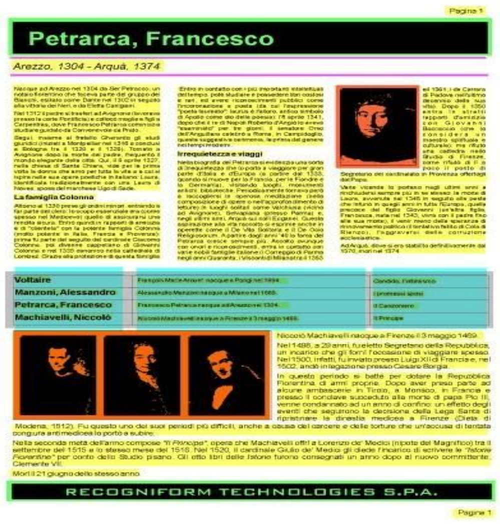


Fig. 5: Output of Page Layout Analysis

Baseline Fitting and Word Detection

Algorithm for text and rows found on Tesseract performs well even in the presence of broken and joined characters speckle noise and page skew. It have the following steps:

1. Connected Component Analysis is performed. Connected component labeling is used in computer vision to detect connected regions in binary digital images
2. For determining text size in a region the median height is used. That components which are smaller than some fraction of the median height mostly being punctuation, diacritical marks and noise are filtered out.
3. The blobs are sorted (into ascending order) using x-coordinate (of the left edge) as the sort key. This sort makes it possible to track the skew across the page.

Once the text lines have been found, the baselines are fitted more precisely using a quadratic spline by a least squares fit. Tesseract does word detection by measuring gaps in a limited vertical range between the baseline and mean line[4]. Spaces that are close to the threshold at this stage are made fuzzy, so that a final decision can be made after word recognition.

Word Recognition

Fig.6 is a block diagram of the word recognizer [6].

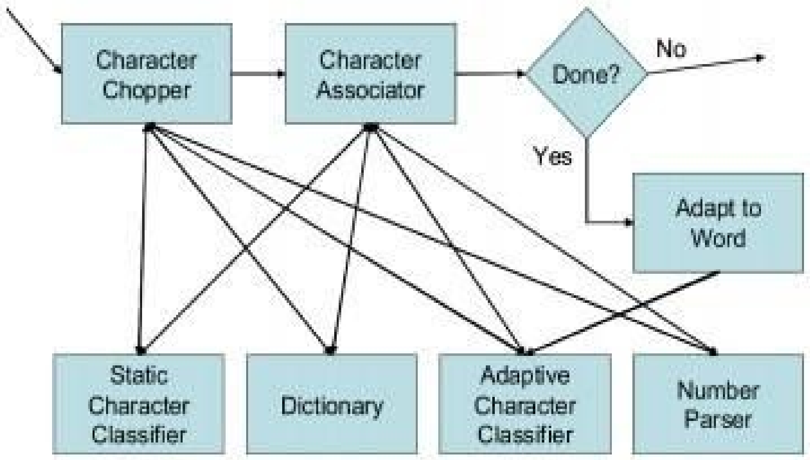


Fig. 6: Block diagram of Word Recogniser.

The word recognizer first classifies each blob, and presents the results to a dictionary search to find a word in the combinations of classifier choices for each blob in the word.

While the word result is unsatisfactory, Tesseract chops the blob with worst confidence from the character classifier. Candidate chop points Fig.7 are found from concave vertices of a polygonal approximation of the outline.

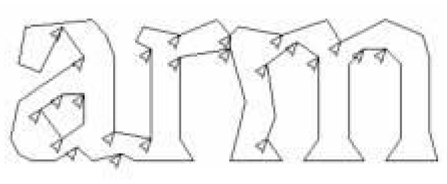


Fig. 7: Candidate chop points

After the chopping possibilities are exhausted, the associator makes an A* (best first) search of the segmentation graph of possible combinations of the maximally chopped blobs into candidate characters [4]. At each step in the best-first search, any new blob combinations are classified, and the classifier results are given to the dictionary again. The output for a word is the character string present in the dictionary that had the best overall distance-based rating.

Static character classification

The features used in classification are the components of the polygonal approximation of the outline of a shape. In training, a 4-dimensional feature vector of (x, y-position, direction, length) is derived from each element of the polygonal approximation, and clustered to form prototypical feature vectors(Hence the name: Tesseract) [6]. In recognition, the elements of the polygon are broken into shorter pieces of equal length, so that the length dimension is eliminated from the feature vector. This process of small features matching large prototypes is easily able to cope with recognition of damaged images. Its main problem is that the computational cost of computing the distance between an unknown and a prototype is very high. To reduce the computing time, in the first step of classification, a class pruner creates a shortlist of 1-10 character classes that the unknown might match using a method closely related to Locality Sensitive Hashing (LSH)[9]. In the second stage, the classifier calculates the weighted distance df of each feature from its nearest prototype as follows [6]:

$$d_f = d^2 + w\theta^2$$

Where d is the Euclidean distance of the feature coordinates from the prototype line and θ is difference of the angle from the prototype.The feature distance is converted to feature evidence Ef using the following equation[6]:

$$E_f = \frac{1}{1 + kd_f^2}$$

The constant k controls the rate at which the evidence decays with distance. As features are matched to prototypes, the feature evidence Ef , is copied to the prototypes Ep . The sums are normalized by the number of features and sum of prototype lengths Lp , and the result is converted back into a distance[6]:

$$d_f = 1 - \frac{\Sigma_f E_f + \Sigma_p E_p}{N_f + \Sigma_p L_p}$$

The computed final distance is used by KNN[10] algorithm for classification.

Adaptive Classification

A more font-sensitive adaptive classifier that is trained by the output of the static classifier is used to obtain greater discrimination within each document, where the number of fonts is limited [8]. Tesseract uses the same features and classifier of the static classifier for Adaptive classification. Since the adaptive classifier learns during the first run, it can only make significantly less contribution near the top of the page if deployed during the first run. Therefore,a second pass is run over the page, in which words that were not recognized well enough in the first run are recognized again.

References

1. Optical character recognition (OCR)
2. Computer vision
3. Tesseract
4. Otsu binarization method
5. Ray Smith. A simple and efficient skew detection algorithm via text row accumulation. In Document Analysis and Recognition, 1995., Proceedings of the Third International Conference on, volume 2, pages 1145–1148. IEEE, 1995
6. Ray Smith, Daria Antonova, and Dar-Shyang Lee. Adapting the tesseract open source ocr engine for multilingual ocr. In Proceedings of the International Workshop on Multilingual OCR, page 1. ACM, 2009
7. Leptonica
8. R Smith. An overview of the tesseract ocr engine. in proceedings of document analysis and recognition.. icdar 2007. In IEEE Ninth International Conference, 2007.
9. Locality-sensitive hashing
10. k-nearest neighbors algorithm (KNN)