

Automation of PostOCR error correction in the digitization of historical texts

Tudor Bumbu, Liudmila Burtseva,
Svetlana Cojocar, Alexandru Colesnicov,
Ludmila Malahov

Abstract

Processing texts from distant historical periods, especially those handwritten in languages with low computational resources, presents significant challenges. Even if modern methods make it possible to achieve, after laborious machine learning procedures, a fairly good rate of correct character recognition, the problem of the correctness of the resulting editable text remains a topical one. This paper presents an approach that contributes to the automation of the PostOCR proofreading process based on the presentation of digitized text using historical fonts, similar to those in the original document.

Keywords: historical fonts, OCR, PostOCR.

Rarity gives things value. What is common is cheap.
What is unique is priceless.

Baltasar Gracian, The Art of Worldly Wisdom (1647)

1 Introduction

Over the years, the interest in the literary-historical heritage is becoming more and more accentuated, and the growing avalanche of scientific works in the field of old documents digitization is a further argument in favor of the concern for revitalizing the heritage, thus ensuring its

accessibility and contributing to the preservation of the original document by widely distributing its digital copy. In the works published by the authors of this article, different historical periods of writing Romanian texts in Cyrillic script have been covered. We moved from the closer (20th century) to the earlier, descending into the depths of the ages, now being at the processing of handwritten documents in the Middle Ages.

When we are dealing with well-preserved documents, behind which there are also rich digital linguistic resources from the historical period in question (ground truth), the problem of obtaining an equivalent editable product and, at the user's wish, transliterated into modern Latin spelling, possibly with updated spelling and vocabulary, can be largely considered as solved.

The challenges arise when working with really old and rare prints or manuscripts, where we are confronted with a much poorer quality of the support on which the text was written or printed, including traces left by the weather or poor conditions of preservation (stains, tears, notes or comments of some readers), transcription errors in the case of manuscripts, lack of standardized spelling rules, writing without intervals or other marks of delimitation (scripta continua). In addition to all this, there was also a large variety of fonts used, for which there were also no standard norms, as we pointed out in [1], illustrating it with the example in Fig. 1 and proposing, at the same time, a method of font classification.

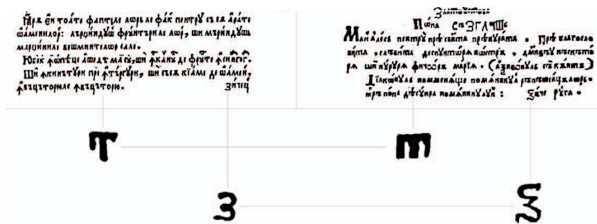


Figure 1. Different presentations of letters in the medieval alphabet

This variety of alphabets and fonts makes the problem of digitizing ancient texts even more complicated. Taking all these aspects into

account, it is natural to follow in the processing of medieval Romanian texts, handwritten or printed in the Cyrillic characters of the corresponding era, some approaches specific to the processing of low-resourced languages.

Broadly speaking, the process of digitizing an old document can be operated through seven main steps, which are implemented in the HeDy platform [2]. One of these steps is the verification of the recognized text, for which the platform offers certain facilities, mainly based on manual intervention by the user.

On the other hand, there is a natural tendency to introduce elements of automation at this step as well, for example, in the identification of post-processing errors. In this paper, we will examine some of the approaches and experiments undertaken for this purpose, specifically based on creating editable text in fonts similar to those of the original image.

2 Historical fonts: problem complexity

Providing appropriate fonts is a necessary part of the historical document digitization, being the essence of the process of restoration of ancient texts and data augmentation for OCR/HTR. Whatever the intended use of digital content, the visualization of the resulting digital content has to be as close as possible to the image of the source document.

The first experiment to verify the correctness of page recognition by comparing the original page image and the recognized text as an image was a kind of naive approach to evaluate the quality of the result using a well-preserved text from the 20th century. Fig. 2 represents a fragment of the original text (Romanian language with Cyrillic script, 20th century) and its equivalent generated with ABBY Fine Reader.

Evidently, fonts differ in size, so the result is predictably low: 57.5% of a similarity, with different letter sizes. After manually adjusting the dimensions (height and width) of the recognized image, the similarity result increased to 99.75% at a resolution of 300 pt.

Obviously, that manual adjustment may not be a suitable solution in the case of historical fonts, which are very different by their nature.

КОМПОНЕНЦА АТЛАСУЛУЙ ЛИНГВИСТИК МОЛДОВЕНЕСК, ВОЛУМУЛ I, ПАРТЯ II

Партя а доуа а Атласулуй лингвистик молдовенеск, волумул I (АЛМ, I, II) инклубе 285 де хэрць. «Консонантизмулуй» ый сынт консакрате 163 де хэрць, «Акцентулуй» ши «Акцидентелор женерале» — 29 де хэрць, яр 3 хэрць де женерализаре редау групаря граюрилор молдовенешть пе база партикуларизацилор вокализмулуй, консонантизмулуй ши феноменелор акцидентале. Ауторул ачестор 195 де хэрць есте *Р. Я. УДЛЕР*. «Морфоложия» есте презентатэ пе 90 де хэрць, консакрате конжугэрий вербулуй ла диферите тимпурь ши модурь, ынтребуинцэрий дубле а пронумелор персонале скурте, формелор де сингулар ши де плурал але субстантивулуй, прекум ши артиколулуй, аджективулуй, нумералулуй, препозиций, адвербулуй ши партикулей. Ла 88 де хэрць аутор есте *В. Ф. МЕЛНИК* ши ла 2 хэрць («Акцум» — 515 ши «Ятэ» — 520) — *В. А. КОМАРНИЦКИ*. Рэспунсуре дин пунктеле 75, 92, 146 ши 236 пентру тоате челе 90 де хэрць граматикале ау фост ынрежистрате де *В. А. Комарницки*.

Деоарече есте ку непутинцэ де а рефлекта пе о хартэ тотализатоаре тоате феноменеле фонетиче, конштиент ам лэсат ла о парте мулте фапте де лимбэ импортанте [де екземплу, ынкидеря луй *е* финал неакцентуат (>*и*), ынкидеря луй *э* финал неакцентуат (>*ы*), тречеря луй *в* (+*о, у*) ла *h* ростиря сунетелор *г* ши *х* ш. а.], прекум ши мулте фонетизме изолате [трей, тот, нор, ырф, фэрэ, (ун) пас, (еу) аш (лукра) ш. а.]. Ши феноменеле ачестя конфирмэ делимитаря унитарцилор локале але териториулуй ностру лингвистик ши комплектазэ характеристика фиэкэруй груп де граюрь.

А трей хартэ синтетикэ аре ла базэ акциденте женерале ши акцентул, фиксате ын кувинте апарте, авынд дрепт скоп комплектаря таблоулуй женерал ал унор субдивизиунь териториале (де екземплу, ал групулуй де граюрь транскарпатиче, ал групулуй де граюрь буковинене ш. а.).

Ын леженделе хэрцилор де женерализаре ын прима колоанэ сынт дате вариантеле фонетиче (але

КОМПОНЕНЦА АТЛАСУЛУЙ ЛИНГВИСТИК МОЛДОВЕНЕСК, ВОЛУМУЛ I, ПАРТЯ II

Партя а доуа а Атласулуй лингвистик молдовенеск, волумул I (АЛМ, I, II) инклубе 285 де хэрць. «Консонантизмулуй» ый сынт консакрате 163 де хэрць, «Акцентулуй» ши «Акцидентелор женерале» — 29 де хэрць, яр 3 хэрць де женерализаре редау групаря граюрилор молдовенешть пе база партикуларизацилор вокализмулуй, консонантизмулуй ши феноменелор акцидентале. Ауторул ачестор 195 де хэрць есте *Р. Я. УДЛЕР*. «Морфоложия» есте презентатэ пе 90 де хэрць, консакрате конжугэрий вербулуй ла диферите тимпурь ши модурь, ынтребуинцэрий дубле а пронумелор персонале скурте, формелор де сингулар ши де плурал але субстантивулуй, прекум ши артиколулуй, аджективулуй, нумералулуй, препозиций, адвербулуй ши партикулей. Ла 88 де хэрць аутор есте *В. Ф. МЕЛНИК*, ши ла 2 хэрць («Акцум» — 515 ши «Ятэ» — 520) — *В. А. КОМАРНИЦКИ*. Рэспунсуре дин пунктеле 75, 92, 146 ши 236 пентру тоате челе 90 де хэрць граматикале ау фост ынрежистрате де *В. А. Комарницки*.

Деоарече есте ку непутинцэ де а рефлекта пе о хартэ тотализатоаре тоате феноменеле фонетиче, конштиент ам лэсат ла о парте мулте фапте де лимбэ импортанте [де екземплу, ынкидеря луй *е* финал неакцентуат (>*и*), ынкидеря луй *э* финал неакцентуат (>*ы*), тречеря луй *в* (+*о, у*) ла *h* ростиря сунетелор *г* ши *х* ш. а.], прекум ши мулте фонетизме изолате [трей, тот, нор, ырф, фэрэ, (ун) пас, (еу) аш (лукра) ш. а.]. Ши феноменеле ачестя конфирмэ делимитаря унитарцилор локале але териториулуй ностру лингвистик ши комплектазэ характеристика фиэкэруй груп де граюрь.

А трей хартэ синтетикэ аре ла базэ акциденте женерале ши акцентул, фиксате ын кувинте апарте, авынд дрепт скоп комплектаря таблоулуй женерал ал унор субдивизиунь териториале (де екземплу, ал групулуй де граюрь транскарпатиче, ал групулуй де граюрь буковинене ш. а.).

Ын леженделе хэрцилор де женерализаре ын прима колоанэ сынт дате вариантеле фонетиче (але

Figure 2. XX century text: source(top) and OCRed(bottom)

Therefore, it is necessary to find a solution for the automatic generation of fonts that are maximally similar to those in the original text.

3 Automating the generation of historical fonts

Automatic font generation is the creation of a new font library based on the reference document images. The main challenge today is to find a good balance between the automation of the process and the quality of the resulting font. The quality measures vary depending on the problem that needs to be solved. The designers require the capturing of font family characteristics, serifs, stroke shapes, etc. Solving our work tasks requires only the maximum proximity of letters between the original image and the generated digital document.

Font generation is implemented in two stages: creation of letters templates and library generation.

Font library generation

Font library generation is still challenging for a large alphabet with complex letters. For European alphabets, which contain a comparably small number of letters, the generation process today is well developed, offering multiple online tools and software like FontLab, FontForge, Calligraphr. For the presented research, we chose FontForge – a free and open-source font editor. FontForge is easy to install on every operating system (OS), being included in standard stores and repositories. It has standalone applications and APIs for the most popular programming languages. FontForge’s main function is to input the letter template, to create the letter, and to place the obtained letter in the corresponding position in the Unicode table of the new font. The letter template can be a ready vector or a raster image (BMP or PNG). The vector letter template is placed directly at the cell, specified by Unicode value (Python example: `glyph = font.createChar(unicodeChar, charName)`). The image letter template initially has to be traced by applying the *autotrace* function. Embedded FontForge tracing is well-trained on European alphabets, but results strongly depend on template image quality. Our tests show that FontForge can trace a clear and strongly binary image

with very high precision, even for complex letters. An example of its application is illustrated in Fig. 3.



Source font (Blackadder ITC Std)	
FontForge generated font Python example: <code>glyph.importOutlines(img_name, background=True)</code> <code>glyph.autoTrace()</code>	

Figure 3. Fancy Font Digitizing Example

Letters templates creation

Automatically creating a letter template is now the main challenge of image-based font generation.

The researchers are employing the language models, which achieve significant results in solving recognition problems. But solving the image-based font generation problem, language models stumble upon the Stroop effect, tending rather to read the text than to analyze the font properties. In the paper [3](March 2025, USA), the researchers tried to avoid the Stroop effect, evaluating different inference strategies, including CoT prompting, MCQ, and few-shot learning. Based on the tests conducted in this paper, the authors stated that the Stroop effect arose due to the modern implementation of attention mechanisms. In their opinion, when solving the recognition problem, the attention mechanism focuses on the edge information of the characters, which pushes the font details into the background. In continuation of our work, we intend to contribute to overcoming this issue and to use the potential of well-trained language models recognition.

Aside from the research, a modern practical solution to the font generation problem involves the creation of letter templates using conventional text recognition software. The main feature of creating letter templates using standard OCR is the need to catch segmentation at the letter level. The APIs of the most popular OCR systems have the

corresponding functions, and the generative machine can answer the prompt: *"Get the coordinate of the text block"*. The problem arises that the researchers are usually restricted by free tools. The majority of OCR tools have access to character segmentation only in commercial versions. Also, the prompt: *"Get the coordinate of all characters in this text block"* is marked as out of free license.

Experiment with the use of FineReader Engine

The second study was performed using the commercial API FineReader Engine 12 (the current version on which all currently used FineReader versions are based). A program (about 200 lines of code) was developed in Python using the *comtypes* and *win32com* interface libraries to cut a handwritten text page image into separate letters. An example of a handwritten page taken from [4] is shown in Fig. 4.

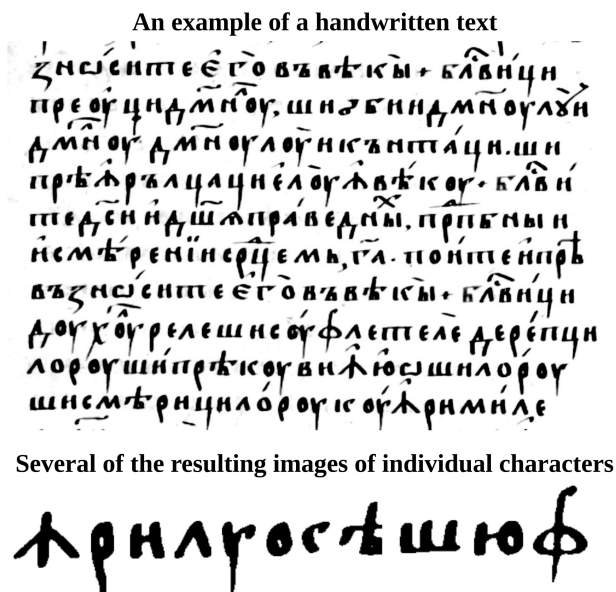


Figure 4. Segmentation by FineReader Engine

To test the efficiency of this method, Russian was set as the recognition language, since the goal was to debug the interaction with FineReader Engine through its API in Python. Using FontForge, a

font was generated, and the original image of the page was compared with the reproduced one, which gave a match of 85%. With FineReader Engine, the training has started to OCR uncial Romanian fonts on the corresponding manuscripts of the 13th-15th centuries.

Perspectives for using the TesseractOCR

In recent (2025) historical documents digitization works, the combined approach that uses both traditional OCR and language models is being researched. Among these works, USA [5], Norway [6], England, and Germany [7] selected TesseractOCR as a traditional OCR tools. Such a choice demonstrates how the free tools requirement inspired the modern revival of TesseractOCR. Despite TesseractOCR's outdated architecture, this software has most of the features demanded by researchers: it is well trained on all rich-resources languages; it works both online and on every operating system; it has an API for the most popular programming languages. For example, in Fig. 3, character segmentation is done by the function `pytesseract.image_to_boxes`. In the case of the historical Romanian alphabets, TesseractOCR training is not time-consuming, since these alphabets differ from the usual Cyrillic alphabet by only a few letters. The TesseractOCR repository has a fine-tuning script that easily adds several characters to the existing model. So, in addition to our tests of FineReader Engine, we tested TesseractOCR for solving our problem.

4 Automated error correction

One of the stages of text processing within the HeDy platform is the verification and editing of the recognized text. Until now, this stage has been performed exclusively manually, with the system providing only support through the display of a virtual keyboard corresponding to the historical period of the document. We can now extend the functionality of this stage by adding the ability to automatically detect OCR errors.

For this purpose, the direct comparison module (DCompars) is included in HeDy as a supporting tool of the PostOCR stage. DCompars performs a direct comparison of two inputs: a preprocessed image of the source document and an image generated from OCR results. This com-

parison is based only on the image features, regardless of the text ones. Focusing on image features allows us to employ the full advantages of modern image processing from powerful libraries of popular programming languages to VLM. The direct comparison method is fully applicable for PostOCR processing of scans of well-printed documents, but has serious limitations in the case of images of historical documents. First of all, modern OCR tools accept source documents that are noisy or have curved text lines. In such cases, the direct comparison method requires additional preprocessing actions, but most of them can be implemented using the same image processing tools. Another limitation is determined by the text content of the input images: only the first mismatch in the text line can be detected. Each incorrectly recognized character shifts the rest of the line, and these mismatches become uninformative.

Despite the limitations, DCompars can significantly support the implementation of error detection and classification. Moreover, this support can be obtained without additional development, because the inputs required for DCompars are generated by existing HeDy tools.

The **first DCompars input** is the result of the source document images' binarization made on the PreOCR stage.

The **second DCompars input** is created from the result of the generation of the full digital copy of the source document images. The full digital copy is generated as a PDF file and supposes the exact mapping of the fonts and layout.

The **DCompars main output** is a set of coordinates of pixels, which differ between two images. This set is organized as a list of contours. The contour contains connected pixels. In addition to an array of coordinates, DCompars output provides bounding boxes of each contour.

At the current stage of development, only part of the DCompars output – the bounding boxes of mismatches – is used in manual error correction to mark the proposed error candidates. But the main planned contribution of DCompars supposed to be the **automatic** correction of errors at the HeDy PostOCR stage.

To demonstrate the potential usability of DCompars, a simple example of manual error analysis is provided. This example shows in

detail how the DCompars application supports the error correction process.

In Fig.5, DCompars inputs are presented: the left image is a fragment of a late 18th-century book; the right image is a synthetic copy of the same fragment, which is generated based on the text obtained by the left image's OCR. This book is printed using a font that was commonly used at this time, so this font is already digitized, named Ponomar, and available for download at several web collections of free fonts. To make the example more informative, the source document image was processed by an advanced binarization module, which is currently in development and is not included in the HeDy pipeline. This binarization removes “the halo” that appears around letters in old books.

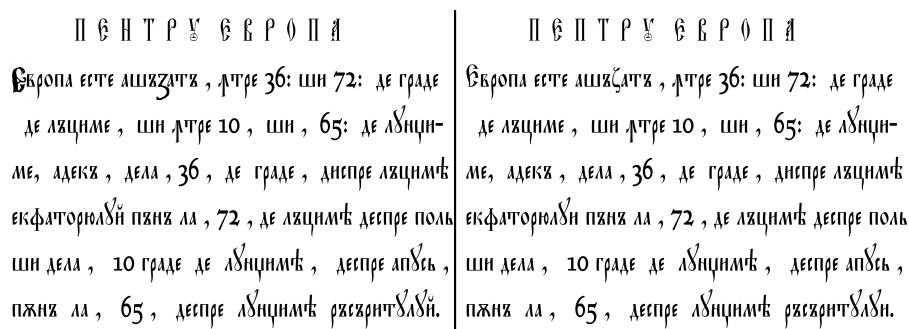


Figure 5. DCompars inputs of the presented example

Fig. 6 shows the output of DCompars for the bounding boxes of mismatches, drawn on the source document image shown in Fig. 5. The bounding boxes are marked by green color, but in the printed paper, the color looks light gray.

The error candidates are analyzed consequently from the top-left corner of the output image.

The **first frame** marks the typical OCR error: the letter **И** is recognized as letter **П**. Such a type of error is usually corrected automatically in the text editor after corresponding training of the embedded corrector. However, despite the ease of correction, in the case of historical documents, this error candidate must be analyzed. It is evident that

The **fourth and the fifth frames** mark the same errors-candidate when letter **Н** is recognized as **Н**. The space between the lines of historical documents often contains many symbols, both informative and not. In most cases, the OCR engine is trained to ignore these symbols. So, the possible reason for this error is the ignoring the upper part of **Н**. But, as mentioned above, untimely OCR stopping is also a possible reason.

Resuming the above example, PostOCR processes, which the application of DCompars outputs supports the most successfully, are automatic error detection and classification.

Taking the automatic error detection and classification as the DCompars planned functionality, the further development of DCompars will focus on the following.

- Adding a tool for transferring bounding boxes in the API of OCR engines. This tool will be helpful for OCR-related error correction, for example, to repeat the OCR of the neighborhood region.
- Adding a tool for analyzing contours using both image and text features to classify error candidates.
- Adding a tool for exporting the output in a form suitable for applications. Since the contour is a vector and is supplied by a bounding box, that allows us to obtain the corresponding raster, there are several PostOCR tasks that can use DCompars output: creating an errors dictionary; generating synthetic data, both ground truth and erroneous; and developing transformer adapters.

5 Conclusions and Further Works

Our research aims to equip the HeDy platform with tools to support the process of identifying and correcting PostOCR errors. The automatic generation of a font library based on the processed document and the inclusion of a module for comparing the original image with the one obtained after OCR (generated using fonts similar to the original ones) allows us to automate error identification, marking their presence in the processed text.

We selected the free TesseractOCR as a backend tool for characters segmentation, due to a simple fine-tuning procedure that allows us to

create a model for historical Romanian alphabets.

Also, in continuation of the presented research, we intend to contribute to overcoming the Stroop effect, which appears during font recognition by well-trained language models, by developing adapters for attention mechanisms.

This paper is the extended and revised version of the conference paper presented at the 5th Workshop on Intelligent Information Systems (WIIS2025).

Acknowledgments. The project SIBIA 011301 “Information systems based on Artificial Intelligence” has supported part of the research for this paper.

References

- [1] T. Bumbu, “Font Classification Model for Romanian Cyrillic Documents,” *Computer Science Journal of Moldova*, vol. 29, no. 3(87), pp. 291–298, 2021.
- [2] T. Bumbu, L. Burtseva, S. Cojocaru, A. Colesnicov, and L. Malahov, *Digitization of Romanian Historical Printings*, Chisinau: Vladimir Andrunachievici Institute of Mathematics and Computer Science, Moldova State University, 2023, 176 p. ISBN: 978-9975-68-497-2.
- [3] Z. Li, G. Song, Y. Cai, Z. Xiong, J. Yuan, and Y. Wang, “Texture or Semantics? Vision-Language Models Get Lost in Font Recognition,” *ArXiv Preprint*, ArXiv:2503.23768, 2025.
- [4] “Psaltirea Voroneteană (I): (mss. 693 dela Bibl. Acad. Rom.)”. [Online]. Available: <https://dspace.bcu-iasi.ro/handle/123456789/22359>, Accessed July 30, 2025.
- [5] S. Backer and L. Hyman, “Bootstrapping AI: Interdisciplinary Approaches to Assessing OCR Quality in English-Language Historical Documents,” in *Proceedings Of The 5th International Conference On Natural Language Processing For Digital Humanities*, 2025, pp. 251–256. [Online]. Available: <https://aclanthology.org/2025.nlp4dh-1.21/>.

- [6] T. Enstad, T. Trosterud, M. Røsok, Y. Beyer, and M. Roald, "Comparative analysis of optical character recognition methods for Sámi texts from the National Library of Norway," in *Proceedings Of The Joint 25th Nordic Conference On Computational Linguistics And 11th Baltic Conference On Human Language Technologies (NoDaLiDa/Baltic-HLT 2025)*, 2025, pp. 98–108. [Online]. Available: <https://aclanthology.org/2025.nodalida-1.11/>.
- [7] G. Greif, N. Griesshaber, and R. Greif, "Multimodal LLMs for OCR, OCR Post-Correction, and Named Entity Recognition in Historical Documents," *ArXiv Preprint*, ArXiv:2504.00414, 2025.

Tudor Bumbu^{1,2}, Lyudmila Burtseva^{1,3},
Svetlana Cojocaru^{1,4},
Alexandru Colesnicov^{1,5}, Ludmila Malahov^{1,6}

¹ Moldova State University, "V. Andrunachievici" Institute of Mathematics and Computer Science, Chisinau, Republic of Moldova

²ORCID: <https://orcid.org/0000-0001-5311-4464>

E-mail: bumbutudor10@gmail.com

³ORCID: <https://orcid.org/0000-0002-9064-2538>

E-mail: luburtseva@gmail.com

⁴ORCID: <https://orcid.org/0009-0003-1025-5306>

E-mail: svetlana.cojocaru@math.md

⁵ORCID: <https://orcid.org/0000-0002-4383-3753>

E-mail: acolesnicov@gmx.com

⁶ORCID: <https://orcid.org/0000-0001-9846-0299>

E-mail: ludmila.malahov@math.md