Vladimir Andrunachievici
Institute of Mathematics and Computer Science

Digitization of Romanian Historical Printings

Tudor Bumbu
Liudmila Burtseva
Svetlana Cojocaru
Alexandru Colesnicov
Ludmila Malahov

Chisinau 2023

CZU 004.9
D 41

VLADIMIR ANDRUNACHIEVICI INSTITUTE OF MATHEMATICS AND COMPUTER SCIENCE, MOLDOVA STATE UNIVERSITY
5, Academiei str., Chisinau, Republic of Moldova, MD 2028
Tel: (373 22) 72-59-82, Fax: (373 22) 73-80-27
E-mail: imam@math.md
WEB address: http://www.math.md

Bumbu, T.; Burtseva, L.; Cojocaru, S.; Colesnicov, A.; Malahov, L.,

Digitization of Romanian Historical Printings.

Recommended for publication by the Scientific Council of the Vladimir Andrunachievici Institute of Mathematics and Computer Science

Descrierea CIP a Camerei Naţionale a Cărţii din Republica Moldova

Digitization of Romanian Historical Printings / Tudor Bumbu, Liudmila Burtseva, Svetlana Cojocaru [et al.] ; Vladimir Andrunachievici Institute of Mathematics and Computer Science, Moldova State University. – Chişinău : [S. n.], 2023 (Valinex). – 176 p. : fig., tab.
Bibliogr.: p. 151-176 (234 tit.). – 70 ex.
ISBN 978-9975-68-497-2.

004.9
D 41

Firma poligrafică „VALINEX" SRL, Chişinău, str. Florilor, 30/1A, 26B, tel./fax 43-03-91,
e-mail: info@valinex.md, http:\\www.valinex.md

Coli editoriale 8,62. Coli de tipar conv. 10,23. Format 60x84 1/16. Garnitură „Times".
Hirtie ofset. Tirajul 70.

Contents

Introduction

Recent advancements in digital technologies, particularly in artificial intelligence, underscore the imperative to automate text digitization processes. These processes play a pivotal role in generating essential resources for the advancement of large linguistic models. Conversely, as early as 2011, the European Commission formulated recommendations on digitization and online accessibility of cultural material, along with digital preservation (source: [1]). The document emphasized the importance of encouraging the development of digitized content from libraries, archives, and museums. It urged member states to bolster their investments in this domain to ensure Europe's continued prominence as a global leader in culture and creative content, leveraging its rich cultural heritage to the fullest extent.

The recommendation has been included as a policy action in many countries (not only in the EU) and a whole industry offering scanning, recognition, and related services has been developed, with the issue of digitization and preservation of cultural treasures.

Large-scale digitization, where the actions are scanning and storing images, began with Project Gutenberg [2], initiated in the 1970s and counting now more than 70000 free access books, later came the Hathi Trust digital library [3] (containing more than 18 millions of printed items), the million-book collection [4] (the goal of digitizing one million books was reached as early as 2007, but work continues in 50 scanning centers) and the Google Books digitization project [5] (where all the books are OCRsed). In addition to these projects, we will also mention the Europeana portal [6], which contains collections of European cultural heritage, including over 4 million texts (books, magazines, articles, etc.) and more.

Even though many documents can be found and read online, they are usually exposed only in image format, and not in machine-readable (or machine-editable) text format. Therefore, the challenge of automating the process of transforming documents into machine-readable (editable) text will be faced by machine learning and computer vision applications.

As an integral part of the Romanian-speaking space, the Republic of Moldova is engaged in the preservation of its heritage through digitization. Several specific problems are inherent in the digitization of the Romanian historical linguistic heritage:

- a large number of periods in the evolution of the language;
- the small volume of widely distributed and available resources;
- the diversity of printing alphabets, in particular, mixed Cyrillic-Latin transitional alphabets;
- the lack of tools for the correct recognition of Cyrillic letters from different historical periods;
- the lack of lexicons suitable for the period of printing of the document;
- demand to transliterate the older Cyrillic scripts to the modern Romanian Latin script.

Therefore, to make our printed heritage accessible as widely as possible, it is necessary not only to limit ourselves to the optical recognition of the characters but also to transliterate the texts into the modern Latin spelling used for the Romanian language. Thus, in the following, by digitization, we mean all the steps involved in transforming a document from a scanned image into two editable documents: one presented in original characters, the other transliterated into Latin characters of the modern Romanian alphabet, and, in some cases, with updated spellings of words.

During document processing, our team used existing software adding self-developed utilities. To integrate different program modules into a unified document processing platform, we proposed an approach called convergent technology [7]. Convergent technology is a general principle of physical and/or logical pooling of resources based on the resemblance, belonging, or similarity of the tasks solved into some interacting software modules. Features of convergent technologies are:

- maximal usage of ready-made solutions of subtasks;

- script language as a glue that gathers different modules together;
- script language to implement new solutions;
- non-intrusive support of modules assembly; modules and scripts interact through external data and signals.

There are two types of documents to be processed: homogeneous (with homogeneous content) and heterogeneous (with heterogeneous content). Homogeneous content consists of text, possibly organized in tables, interspersed with non-textual elements kept as images. Let us note that many non-textual content elements can be presented in a scripting language. The main features of the heterogeneous content are as follows [8]:

- the document is not exclusively in natural language;
- there are scripting languages presenting their components;
- the graphic representation can be re-rendered from the scripts.

Therefore, we will differentiate textual content, images, and script presentable content inside a heterogeneous document. Content that can be presented as a script should be recognized; the most obvious advantage of script presenting are possibilities to search and modify it. A good example of a heterogeneous document is an encyclopedia.

This book examines the problems of digitization of both types of documents - homogeneous and heterogeneous, printed in the Roman language with Cyrillic characters. It is based on the publications of seven years of research and development by our team in this field, discusses the current state, challenges, and perspectives, and is structured into seven basic chapters.

Chapter 1 describes the history of Romanian writing and printing, and encircles the set of processed documents: printed documents and books including historical ones from the 17th century up to the 20th century. The scripts in the documents are mostly Cyrillic.

In Chapter 2, we analyze the tools for image preprocessing paying special attention to methods of improving the quality of old document images.

The next Chapter 3 includes a review of methods, tools, and resources for the digitization of documents of historical and cultural heritage for different languages and periods. A special section is dedicated to the work done for the processing of Romanian texts printed with the Cyrillic alphabet.

Chapter 4 summarizes the experience with documents from the 17th

century. It grounds our approaches to the design of the technology for processing historical texts (printed in Romanian in Cyrillic script in the 17th century), describes the methods developed, and argues for the use of some of the existing modules. The process of developing OCR models is presented, and the neural network-based font classification method is described.

We have devoted a separate Chapter 5 to the processing of texts printed with transitional alphabets, considering that this subject deserves special attention because of the variety of alphabets (with at least 17 attested), characteristic of the first half of the 19th century.

Chapter 6 describes the application of transliteration, highlighting problems specific to each historical period, the most difficult in this respect being the writing of 20th-century texts in modern Cyrillic script.

The next Chapter 7 discusses IT support of work with heterogeneous documents which usually contain not only text but also various content types like mathematical and chemical formulas, musical scores, diagrams, technical drawings, chess notations, etc.

The technology needs the use of an integrated Web platform the structure and functionality of which are described in the last Chapter 8.

Acknowledgments. The authors express their deep gratitude to their colleagues from the Laboratory *Programming Systems* of the Vladimir Andrunachievici Institute of Mathematics and Computer Science under the State University of Moldova for their collaboration, support, and valuable discussions. We would like to express our acknowledgment to the National Agency for Research and Development of the Republic of Moldova for funding project 20.80009.5007.22: *Intelligent information systems for solving ill-structured problems, processing knowledge and big data* within the framework of which the work was carried out in the period 2020-2023.

Chapter 1

The historical evolution of Romanian writing and printing

1.1 Stages and timeline

After the invention of movable type printing by Gutenberg around 1445, this technology quickly spread throughout Europe. The first book on the territory of modern Romania was printed in Târgoviște in 1508 in Church Slavonic. The earliest surviving printed publication in the Romanian language is the 1561 Gospel Book [9]. Fig. 1.1 shows two pages from *The Romanian Apostolos* printed by Coresi in Brașov in 1563.

The subsequent development can be described as a combination of three processes.

Firstly, there was continuous evolution in the Danubian Principalities and later in Romania. According to [10], it was in the late 15th and early 16th centuries that the process of replacing Church Slavonic with Romanian began in administrative and governmental structures as well as in the church. From Church Slavonic, the Romanian language adopted the Cyrillic alphabet. The letter Ѫ is unique to the Romanian alphabet among other Cyrillic-based alphabets.

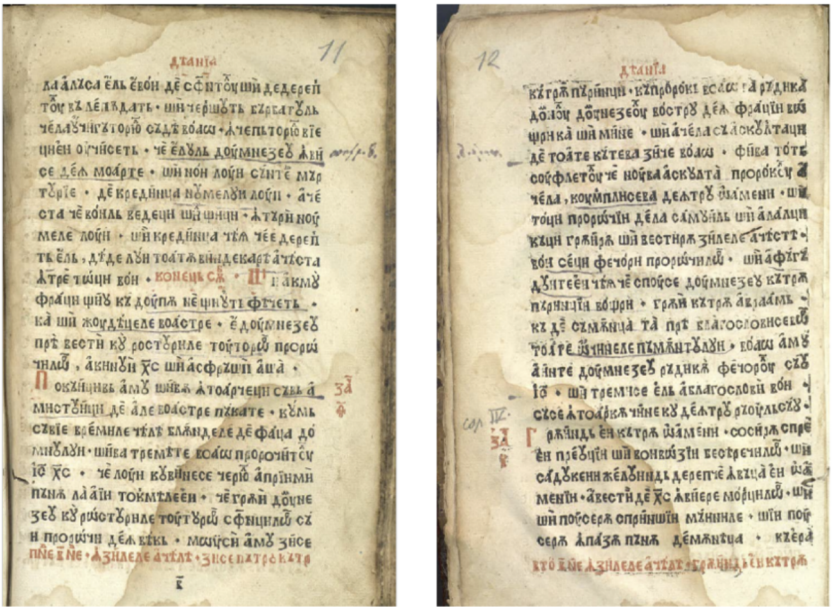


Figure 1.1: Pages from *The Romanian Apostolos*, Braşov, 1563.

The division of text into words can already be observed in the 1561 Gospel Book. By the late 18th century, there was a simplification of the script: a transition to Arabic numerals, the abandonment of abbreviations, and diacritical marks. This was followed by the shift to the Latin-based script through transitional scripts, accompanied by the abandonment of uncial forms. In 1904, the simplification of the Latin script of 1860 led to the modern script. These five stages are reflected in rows 1–5, Tab. 1.1.

Secondly, when considering detailed data from Bessarabia, the situation becomes more complicated. This region became a part of the Russian Empire in 1812, joined Romania in 1918, and then switched from Romania to the USSR twice. The Romanian speaking regions on the left side of river Dniester (Transnistria) remained within the USSR and even held autonomous status there for a period. In Bessarabia under USSR and Transnistria, the MC variant (Tab. 1.1, row 6) of the Romanian script was used.

Table 1.1: Evolution of Romanian scripting.

1	**RC**	The Romanian Cyrillic script	Up to 43+ letters, imitating manuscripts.
2	**SRC**	The Simplified Romanian Cyrillic script	In the uncial and civil variants (see Tab. 1.2).
3	**TR**	The TRansitional scripts	Civil SRC with several Latin letters
4	**RL**	The Romanian Latin script	The initial variant.
5	**MRL**	The Modern Romanian Latin script	Mainly phonetic script.
6	**MC**	The Moldavian Cyrillic script	*Ad hoc* transcription of Romanian sounds by Russian letters.

The detailed timeline of this evolution, including in Bessarabia and Transnistria, is shown in Tab. 1.2.

Table 1.2: Timeline of the Romanian scripts.

From	**1560**	**1760**	**1812**	**1830**	**1862**	**1904**	**1918**	**1924**	**1932**	**1938**	**1940**	**1941**	**1944**	**1989**	**1992**
Romania	RC	SRC[1]	SRC[1]	TR	RL	MRL	MRL				MRL	MRL	MRL		
Bessarabia			SRC[2]								MC		MC	MRL	MRL
Transnistria								MC	MRL	MC					MC

[1]Uncial variant (in Romania).

[2]Civil variant (in Bessarabia, church publications). See details in Sec. 1.3 on p. 16.

Lastly, irregular attempts at using the Latin alphabet occurred long before its official adoption in 1860. Several Romanian books were printed in Latin script with varying orthographic models (Germanic, Hungarian, Polish) [11, p. 201-248].

This intricate history reflects the dynamic interplay of linguistic, cultural, and geopolitical factors that have shaped the trajectory of Romanian writing and printing. See portions of texts in all six regular Romanian scripts in Fig. 1.2 on p. 13.

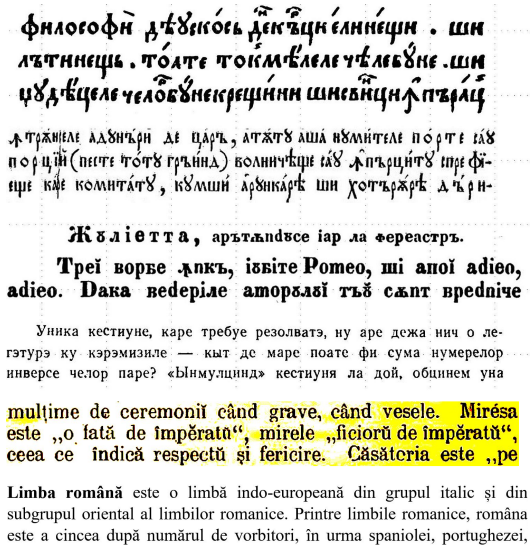


Figure 1.2: Six regular Romanian scripts

1.2 The Romanian Cyrillic script (RC)

In old grammars of the Romanian language, a Cyrillic alphabet of 43 letters is presented. We reproduce it in Tab. 1.3 on page 14, following source [12].

The order of the letters varies in different sources. As for their quantity, it depends on what is considered a letter. In the alphabet shown, for example, the letter Ѕ was used only for writing numbers. The letter Ы was always included in the alphabet but was not used in word spellings. Conversely, Й was not included in the alphabet as it was considered a variant of the letter И with a diacritic mark.

Romanian Cyrillic uses a variety of diacritic marks (Tab. 1.4 on page 15). Example:

RC: Ѧдевѣ́рь шѝ но́и ѡ́амени сѫ́нтемъ́ шѝ ѣ́м пѹ́тѹ́т шѝ грешѝ

3 1 2 1 31 1 5 2 4 15 2 2

MRL: Adevăr și noi oameni suntem și am putut și greși

Eng.: Truth be told, we are human, and we can commit sins

Table 1.3: The Romanian scripts.

	RC, SRC	TR	RL, MRL	MC		RC, SRC	TR	RL, MRL	MC
1	Аа	a	a	а	22	Оуѹ	ȣ	u	у
2	Бб	б	b	б	23	Фф	f	f	ф
3	Вв	в	v	в	24	Хх	x	h	х
4	Гг	g	g, gh	г	25	Ѡѡ	o	o	о
5	Дд	d	d	д	26	Цц	ц	ţ	ц
6	Ее	e	e	е	27	Чч	ч	c (before e, i)	ч
7	Жж	ж	j	ж	28	Шш	ш	ş	ш
8	Ѕѕ				29	Щщ	щ	şt	шт
9	Зз, Ꙁꙁ	z, ḑ	z, ḑ (RL) z (MRL)	з	30	Ъъ	ъ	ă	э
10	Ии	i	i	и	31	Ыы			
	(Йй)	ĭ	ĭ (RL only)	й, ь	32	Ьь			
11	Її	i	i	и	33	Ѣѣ	ea	ea	я
12	Кк	k	c, ch	к	34	Юю	iȣ, ĭȣ	iu	ю
13	Лл	l	l	л	35	Ѫѫ	î	â, î	ы
14	Мм	m	m	м	36	Ꙗꙗ	ia	ia	я
15	Нн	n	n	н	37	Ѧѧ	ia	ia	я
16	Оо	o	o	о	38	Ѱѱ	ps	ps	пс
17	Пп	п	p	п	39	Ѳѳ, Ꙋꙋ	t, ft	t, th	т
18	Рр	p	r	р	40	Ξξ	ks	x	кс
19	Сс	s	s	с	41	Ѵѵ	i, ȣ	i, v	и, в
20	Тт	t	t	т	42	Ѧ҃	în, îm	în, îm	ын, ым
21	Уꙋ	ȣ	u	у	43	Ѫ҃	ѫ	g (before e, i)	ж (until 1967), ж

RC followed Churchslavonic number notation that is shown in Tab. 1.5 on page 15. Numbers up to 999 are written by letters with a specific overline mark.

Unlike the Romanian version, in Church Slavonic the number 800 was written ѽ. The letter ѿ was not part of the Romanian alphabet, although it may be used in Romanian church texts.

When forming compound numerals in the range 11−19, the lowest digit was written first, followed by і (10). The rest of the numerals were written from the highest digit to the lowest: compare ві= 12 and ка= 21. This is explained by the fact that in Romanian, as well as in Church Slavonic, in the verbal recording of numerals, the order of the digits is exactly the same.

Numbers bigger than 999 can be written in two manners: with underline symbol ҂ for thousand, or framed, as shown in Tab. 1.5.

Table 1.4: Overline marks in the RC script.

1	и́	Usual stress
2	ѝ	Stress at the end of the word
3	и̓	Vowel is the first letter in the word
4	и̓́	Both the first vowel and stress (stress may be omitted)
5	м́	Mute letter ь omitted after the consonant at the end of the word
6	й	Forms Йй from Ии
7	ӣ	Abbreviations and numbers

Table 1.5: The Churchslavonic number notation.

$1 = \vec{а}$	$10 = \vec{і}$	$100 = \vec{р}$	Examples	Framed notarion
$2 = \vec{в}$	$20 = \vec{к}$	$200 = \vec{с}$	$12 = \vec{в}\mathrm{і}$	$3000000000 = \Xi\overset{+}{\underset{\text{III}}{\Gamma}}\Xi$
$3 = \vec{г}$	$30 = \vec{л}$	$300 = \vec{т}$	$123 = р\vec{к}г$	$200000000 = \boxed{\text{В}}$
$4 = \vec{д}$	$40 = \vec{м}$	$400 = \vec{у}$	$1234 = \text{„}ас\vec{л}д$	$10000000 = \overset{+}{\underset{+}{\text{А}}}$
$5 = \vec{е}$	$50 = \vec{н}$	$500 = \vec{ф}$	$12345 = \text{„}\vec{в}\mathrm{і}\ т\vec{м}е$	$3000000 = \dot{\Gamma}$
$6 = \vec{ѕ}$	$60 = \vec{ѯ}$	$600 = \vec{х}$	$123456 = \text{„}р\vec{к}г\ у\vec{н}ѕ$	$200000 = \hat{\text{В}}$
$7 = \vec{з}$	$70 = \vec{о}$	$700 = \vec{ѱ}$	$1234567 = \text{„„}\vec{а}\ \text{„}с\vec{л}д\ ф\vec{ѯ}з$	$10000 = \text{Ⓐ}$
$8 = \vec{и}$	$80 = \vec{п}$	$800 = \vec{ѡ}$	$12345678 = \text{„„}\vec{в}\mathrm{і}\ \text{„}т\vec{м}е\ х\vec{о}и$	
$9 = \vec{ѳ}$	$90 = \vec{ч}$	$900 = \vec{ц}$	$123456789 = \text{„„}р\vec{к}г\ \text{„}у\vec{н}ѕ\ ѱ\vec{п}ѳ$	

RC abbreviations are of four kinds: taking letters above the line with a superscript marking, the same without a marking, omitting individual letters with a superscript marking, and abbreviating some frequently occurring words. Examples from [13]: $\widehat{\text{а}}\text{тє} = \text{а}\breve{\text{с}}\text{тє}$, $\text{д}\overset{\text{м}}{\text{у}}\text{нєзи́}\overset{\text{у}}{\text{д}} = \text{д}\overset{\text{у}}{\text{м}}\text{нєзи́}\overset{\text{у}}{\text{д}}$, $\text{сфи́нтж} = \text{сфи́нтж}$, $\text{І}\vec{\text{с}}\vec{\text{х}}\text{с} = \text{І}\text{ис}\overset{\text{у}}{\text{х}}\text{с}$ Христо́с (Jesus Christ). There are also combinations of these abbreviations, e.g. $\text{пр}\widehat{\text{ѣ}}\text{сло́вїє} = \text{пр}\text{ѣдосло́вїє}$ (bringing the letter up with a mark and omitting the letter).

1.3 The Simplified Romanian Cyrillic script (SRC)

At the last third of the 18th century the RC script was simplified by omitting overline marks, revealing abbreviations, and use of the Arabic numerals. In Romania, it kept uncial letter form. Example from [14, p. 10]:

> Дрептатеѫ дн аритметикѫ сѫнт доѹж операціи фондаментале, адекѫ Ѧдиціа шн Сѹбтрацерѣ; шн алте доѹж каре сѫнт нѹмаи ѹн каз партикѹлар ачелор доѹж диитъй, шн сѫ нѹмеск мѹлтипликаціа (дмѹлцирѣ) шн дивизіа (дмпърцирѣ.)

In Bessarabia, the local church administration was a single actor that regularly published in Romanian, including periodicals. These publications used the civil variant of the SRC script. Non-church publishing was rare and followed the script actual at the moment in the territory of Romania. This continued until 1918. See details in [15].

1.4 The Transitional scripts (TR)

The Transitional scripts (TR) aimed to move the Romanian writing to Latin-based civil script step by step, replacing several letters each time. That is, SRL was the starting point that was rebuild in civil (non-uncial) design, and then more and more letters were replaced by Latin ones.

In reality different typographies replaced fonts discordantly; supposedly, the replacement was performed while renewing worn letterpunches. As a result, more than dozen variants of the TR scripts are counted.

See details in [11].

1.5 The Romanian Latin script (RL)

The Romanian Latin (RL) script was introduced in 1860 in Romania, and in 1862 adopted in Transilvania and Bucovina that were then under foreign sovereignty.

The script was introduced by the order of the minister of internal affairs Ion Ghica on February 8, 1860 [11, p. 332—334]. The order was very short

containing the letter-to-letter table of the actual transitional and new Latin-based scripts, and several simple rules in plus.

Some peculiarities of the introduced script were: variants of letters (**ţ**, **ts** for **ц**, and **ş** and **ss** for **ш**); acute accent for combinations of vowels (**é** for **ea**, **ó** for **oa**); **ḑ** for sound /ʤ/ that was replaced slightly later by **ḍ**; two letters (**â**, **î**) for sound /ɨ/; mute **ŭ**, etc. See [11, p. 333].

In the period 1860—1904 some authors proposed and apply even more complicated etymological orthography. For example, they wrote **ç** in place of **ţ** because the Latin source word contained **c**, or used up to five letters (**â**, **ê**, **î**, **ô**, **û**) for /ɨ/ depending of the corresponding vowel in the Latin prototype word, etc.

1.6 The Modern Romanian Latin script (MRL)

In 1904 the RL script was replaced by the Modern Romanian Latin (MRL) script that is used till nowadays. MRL fixes the (almost) phonological Romanian orthography.

The phonological principle concerns the correspondence between letters and phonemes or typical sounds. In a phonological writing system each phoneme is denoted by a letter and each letter renders in writing a single sound.

Since 1904 MRL got minor changes in 1932, 1953, 1964, and 1993 [16]. The current variant (1993) was subsumed in 2005 by the normative dictionary [17].

The following example of the MRL script is taken from the Romanian Wikipedia [18], article *Limba română* (The Romanian language):

> Limba română a evoluat din latina orientală, contactul prelungit cu populațiile slave fiind la originea unei părți importante din vocabular. În evul mediu și în perioada premodernă au intrat în limbă un număr limitat de cuvinte maghiare, turcice vechi, turcice otomane și grecești. O influență puternică a avut-o limba franceză în secolul al XIX-lea.

As an exclusion of phonological principle, MRL contains two letters **Â** and **Î** for just one sound /ɨ/. In 1953—1964 only **Î** was used, and in 1964—1993

Â was restored only in the word **român** (Romanian) and its derivatives. The restoration of the Latin-based alphabet in MSSR took place in 1989 that fixed for the Republic of Moldova the actual in 1989 Romanian variant with restricted usage of **Â**. This variant remains official until present in the Republic of Moldova while Romania restored **Â** fully since 1993. In practice, only schools and official editions in the RM follow these restrictions for **Â**; they are ignored by most people and non-governmental publishers.

1.7 The Moldavian Cyrillic script (MC)

The Moldavian Cyrillic (MC) script was created with the creation of the Moldavian Autonomous SSR in Transnistria in 1924.

Moreover, since 1920s till 1950 a theory ruled in the USSR that language is class specific.

The initial period of MC writing (1924–1951) is associated with an extremely specific lexicon. It is characterized by: use of Russian words; deletion of Romanian words that were claimed as "bourgeois"; introduction of self-invented neologisms for abstract notions that cannot be found in the local dialect; fixing of local dialect peculiarities as the language norm.

The MC was practiced in the MASSR until 1932. In 1932 Transnistria was switched to the Romanian language norms and the MRL script. The year 1938 brought the next transfer back to MC. In 1940 Romania returned Bessarabia to the URSS. Moldavian became the state language of the newly created Moldavian SSR that included Transnistria but excluded Budjak.

Since 1941 until 1944 Bessarabia and Transnistria was administrated by Romania. Since 1944 the MSSR was restored.

In 1950 the Soviet leader Josef Stalin published his work *Marxism and problems of linguistics*, which dethroned the theory of the class specific languages. This permitted to reorient MSSR to the linguistic standards of Romania, keeping nevertheless the MC script. Finally, in 1967 MC was extended by the letter **Ӂ** for the sound /d͡ʒ/.

MSSR got independence as the Republic of Moldova in 1991, while the MRL writing was established in 1989. The enclave of Transnistria uses MC until present.

The following citation from [19, p. 14] gives an example of MC writing:

> Пентру а не конвинже де ачаста, сэ анализэм кыт де мулт пот фи мутате кэрэмизиле спре дряпта. Дакэ ну авем ла ындемынэ кэрэмизь, сэ луэм доминоул сау, ын чел май рэу каз, ун симплу клит де кэрць. Кондиция принчипалэ есте кондиция де екилибру. Не вом стрэдуи сэ ынцележем ын че констэ еа.

Chapter 2

Tools for image preprocessing

2.1 The importance of image quality in OCR

It is well-known that the quality of an image significantly influences the accuracy of optical character recognition (OCR). Historical documents are particularly vulnerable to quality degradation. Factors such as humidity, light exposure, physical wear and tear, and even certain writing materials can lead to fading, smudging, or the deterioration of the text over time. Such damage can introduce noise and other distortions, making it challenging for OCR systems to identify and translate the characters correctly. Thus, preprocessing the images from documents, especially older ones, to enhance their qualitative appearance becomes a crucial step in the digitization workflow. Additionally, preprocessing scanned documents is a vital step in machine learning, as during this phase the initial data is adapted to serve as a compatible input for an OCR system.

Considering our interest in extracting text from images, we will emphasize those image preprocessing procedures that ensure the highest possible OCR accuracy. In this chapter, we will describe the nuances of using certain tools for image processing—namely Scan Tailor[20] and ABBYY FineReader(AFR)[21].

2.2 Preprocessing techniques for scanned documents

Preprocessing techniques can rectify many common issues found in deteriorated or low-quality scans. For example:

• *noise reduction* technique removes specks, dots, or unwanted artifacts, ensuring that they are not mistakenly recognized as characters;

• *binarization* converts the image to black and white, making the black text clearer against the white background;

• *skew correction or orientation correction* makes the text to appear straight if a document is scanned at a slight angle, so it aligns correctly for the OCR system;

• *resolution enhancement* can provide finer details of the text, especially for older documents where characters might be closely spaced;

• *character thickening* is akin to the resolution enhancement technique previously mentioned – just as enhancing resolution can provide finer details of the text, thickening thinned-out characters (some characters may be thinned-out by binarization) ensures they are recognizable and accurately transcribed by OCR systems.

The binarization is achievable in all the tools described next. In AFR it appears as a default option when setting up the image processing settings. Therefore, we can infer that FR provides some necessary options for preprocessing old texts. However, it does not encompass the entire spectrum of required tools, necessitating the involvement of additional instruments. There is a wide range of these tools available, including some that are freely distributable, such as Scan Tailor, OpenCV and GIMP. One of the essential preprocessing modules for old documents that is absent in FR pertains to character thickening. Much like rust corrodes iron, wear and tear specific to books can thin out characters. Furthermore, certain binarization methods can reduce the thickness of the lines in characters. To thicken them again during the image preprocessing phase, we use a specialized module in Scan Tailor, a tool that we will describe in further detail below.

2.3 Preprocessing with Scan Tailor

Scan Tailor is an interactive preprocessing tool for scanned pages. It performs operations such as page splitting, deskewing, adding/removing borders, and others. It is a Free Software written in C++ with Qt and released under the General Public License version 3 [20].

This tool has proven to be quite suitable for our cases considering several factors, including processing speed and a default document processing order that is accessible to most users. Therefore, in the following, we will focus on applying the Scan Tailor software and examine the steps of image processing with this software.

For a document consisting of multiple pages, it is necessary to process each page separately. Most often, we use this tool for binarization, to clean the image of unwanted artifacts (spots, blurriness with erased areas, noise represented by a multitude of black dots next to characters, etc.), to straighten the rows, to thicken the characters and to improve the resolution. If several pages are in a single image, then we will apply the split pages step. The type of separation is determined automatically, but it can be set manually and can be applied to all pages simultaneously or to individual pages.

An important module we utilize through Scan Tailor is binarization. It is defined as a function that maps the intensities of the input image pixels to values of 0 (black) or 1 (white) in the binarized output image. There are two main classes of binarization methods. The first determines a global threshold in the image, simply a grayscale level, and then assigns the value 0 to all pixels with a value less than this threshold and the value 1 to the other pixels. The most used method in this class is the Otsu's Method[22]. The second class consists of a set of methods with a local threshold. It determines a different binarization threshold for each pixel, rather than using a global threshold for the entire image. Local threshold methods can use various techniques to determine the binarization thresholds for each pixel. Some may consider information about the near neighbors of the pixel to determine the threshold, while others might use global statistics to find a threshold that adjusts at the local level. After determining the binarization thresholds for each pixel, they are used to decide whether each pixel should be treated as being white or black in the final binary image. In Scan Tailor, binarization is implemented

by equalizing the illumination based on the study [23], smoothing via the Savitzky-Golay filter[24], followed by the actual binarization based on Otsu's Method, and, ultimately, the removal of broken edges. The removal of these broken edges involves using a template image as a reference to locate and eliminate discontinuous margins from the input image. In this context, the template image would represent a linear edge without interruptions, and the algorithm would search for this in the input image, replacing any broken edge with an edge similar to the one in the template. Our experience indicates that it's beneficial to save documents in a "black-and-white" format since this has shown better accuracy in OCR. However, opting for this approach demands special attention, as discoloration might lead to the loss of some text elements. This loss can be compensated to some extent by thickening the characters (an option selectable upon saving), but it's crucial to test on a few pages before applying the procedure to the entire document.

One of the crucial modules at the preprocessing stage is the one that corrects image resolution. Resolution represents the number of pixels present in an image and is vital for OCR models to work efficiently. Resolution can vary depending on when the document was printed, its wear state, and the quality of the scanned image. To achieve optimal results, it's essential to adjust the images' resolution before OCR.

To obtain optimal results during the processing of old documents, it is essential to consider the necessary resolution for each period. For instance, 17th-century documents require a resolution higher than 900DPI to detect ligatures, but below 1300DPI to avoid detecting multiple characters together (see Fig. 2.3, p. 25). The sufficient resolution for 18th 19th and 20th-century documents is 300DPI, also referred to as the *gold standard of resolution.* If an image with a different resolution than the images used for OCR model training is submitted for recognition, issues or anomalies might occur during recognition, especially for 17th-century documents. This is because, instead of detecting and segmenting the entire character, only a character portion or multiple characters and rows might be segmented together (see Fig. 2.1 on p. 24, Fig. 2.2 on p. 25, Fig. 2.3 on p. 25 taken from a 17th-century printed document with characters from the Romanian Cyrillic alphabet.

Detecting a character doesn't mean recognizing the character. Actual

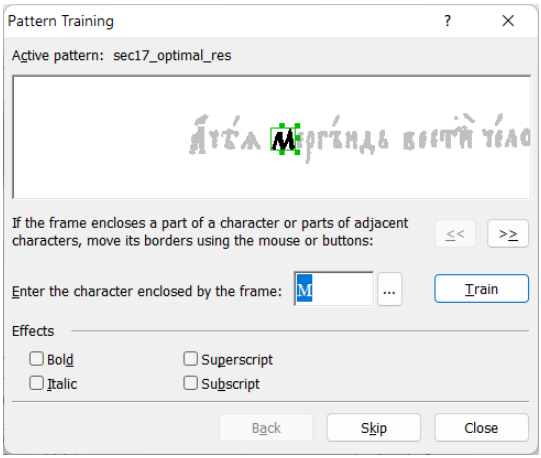


Figure 2.1: Detection and segmentation of characters in an image with optimal resolution

character recognition follows after detection and segmentation). It is crucial to account for this resolution difference to obtain optimal results during document processing.

Resolution setting modules in Scan Tailor and OpenCV allow manual resolution setting by the user. By default, Scan Tailor sets the output image resolution to 600DPI. We mentioned *output image resolution* because there's also an input image resolution parameter, meaning that before Scan Tailor starts other image processing, the loaded image should already have an appropriate resolution. For instance, applying the noise removal on a 50DPI resolution image might have no effect in Scan Tailor, hence sometimes input resolution setting is required. The input resolution parameter or simply resolution in Scan Tailor is set to 600DPI by default.

The preprocessing with Scan Tailor is a supervised process. With every preprocessing option chosen, a specialist needs to adjust the parameters and initiate each process individually. As a result, the preprocessed images are stored in a designated folder, named "out" in TIFF format. These images are ready for OCR.

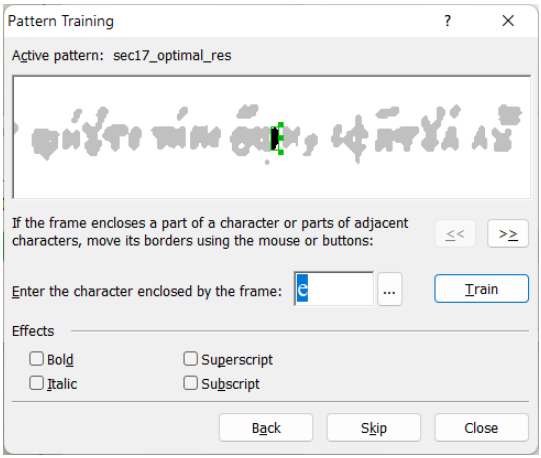


Figure 2.2: Detection and segmentation of characters in an image with too low resolution (96DPI)

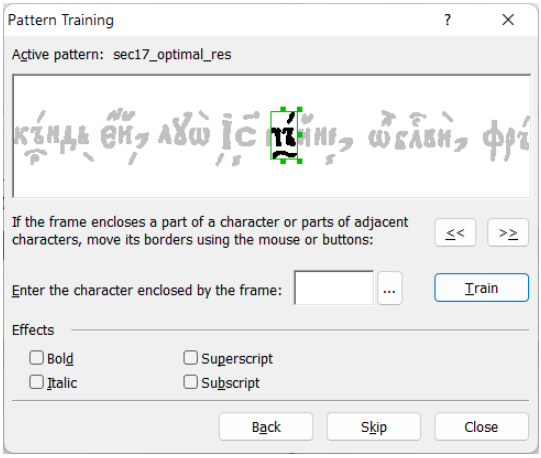


Figure 2.3: Detection and segmentation of characters in an image with too high resolution (2000DPI)

Chapter 3

OCR solutions across languages and their evolution into integrated recognition platforms

3.1 The comparative description of OCR tools for processing historical documents

Various OCR methods and tools have been proposed for processing historical documents; however, while some excel in certain tasks and are mostly commercial products, others - free of charge - offer comparatively reduced functionality in terms of character recognition accuracy but still present suitable tools for research and experimentation.

Optical Character Recognition for modern printed documents using the Latin alphabet works very well (with over 99% accuracy in most cases) and is often considered a solved problem [25]. Traditional OCR methods available in commercial and free software products operate as follows: during the OCR process, an image of a printed page is segmented into characters, which are then compared with sets of abstract features describing examples of

characters previously learned from a character set of a font. The similarity between the learned and recognized characters, the clear separation of black characters from the white background, and the modern spelling of words contribute to excellent recognition results.

Until 2014, historical documents posed a severe limit to the effectiveness of OCR, as most available OCR engines had been extensively trained with modern fonts. However, since historical fonts are very different from modern ones, they require separate training by specialist teams. For relatively old texts (19th century), the OCR results from commercial engines are often less satisfactory [26], [27]. Even Antiqua fonts (Roman glyph forms) of historical prints often lead to a character recognition accuracy of about 85%, being recognized with ABBYY FineReader, a leading commercial product [28]. Since 2019, ABBYY has implemented new artificial intelligence algorithms[29], which substantially contribute to the recognition of languages based on complex scripts, such as Chinese, Japanese, and Korean. Hence, different models are used for Latin, Cyrillic, Arabic, and other scripts. For example, for Arabic script, an end-to-end approach is used for word recognition without character separation. A special architecture, combining convolutional and recurrent neural networks, solves this task. For old Latin and Cyrillic scripts, a mixed approach is used, switching between different recognition models based on the visual quality of the text. This significantly improves both the speed and accuracy of optical character recognition, including from historical documents[29].

There are two well-known methods of training OCR models in this field: training on synthetic data (images generated from existing electronic text and computer-available fonts); or training on real data (pairs formed from glyph or character images and their transcription - the Unicode character)[30]. The first method avoids the need to generate ground truth data necessary to establish the link between glyph forms and Unicode characters during training and also does not require the preprocessing of real images. As the entire training process can be automated, this training method is preferred whenever applicable. However, the multitude of historical fonts cannot be matched with existing fonts, and irregularities in word spacing, inferior quality of scanned images, wear stains, etc., in such documents lead to inferior

recognition results compared to training on real data [28]. OCR training for historical documents must therefore rely on a process that uses real data, meaning that training examples taken directly from printed documents become a key resource. Historical corpora with such examples, which could serve as training data for historical fonts, are not yet available in sufficient quantity (as mentioned above, only small groups of specialists are concerned with this). Another issue is variability. Old typographies are characterized by a greater diversity of fonts, as the process of designing, producing, and distributing metal letter types had not yet become a separate profession, and old typographies had to produce their own sets of letters, leading to a wide variety of historical fonts [31], [32]. Therefore, it is problematic to train associations between printed glyphs and the UNICODE characters they represent, using data from one typography and applying it to another.

In the paper [28], the authors note two issues regarding the application of OCR methods to historical printed documents. First, an individual model must be trained for a specific book with its unique typography. This can be done by transcribing a portion (one or more pages) of the printed document's text, usually requiring linguistic knowledge. Second, even if this model works well for the book it was trained on, it normally does not yield good OCR results for other books, even if their fonts look similar. This typographic barrier must be overcome to effectively use OCR methods in building a historical corpus. The authors then describe experiments addressing both issues. Firstly, they outline a procedure for training individual models, using a new recognition algorithm based on recurrent neural networks, as implemented in the *OCRopus OCR* engine [33]. As training material, the authors use scanned documents from the RIDGES corpus[34]. The individual model is trained on a single document with its specific fonts and thus optimally adapted to this book. These models produce excellent recognition results for unseen (unprocessed) pages of the books they were trained on, but unfortunately yield mostly poor results when applied to any other documents with even slightly similar fonts. Next, the authors explore the viability of training mixed models on a range of different typographies, pooling training material from a variety of books in the hope that these models are better able to generalize the recognition process for books from which character examples were not

taken in the training set.

Many studies on the recognition (OCR) of historical documents have largely focused on *Tesseract* [35], an OCR engine that can be trained on artificial images generated from computer fonts. Tesseract is undoubtedly one of the most popular open-source OCR engines. It was created at Hewlett-Packard between 1985 and 1994 and was made open source in 2005. Tesseract is still under active development by Google and modern versions includes a new engine based on neural networks with an recurrent architecture. Both engines have strengths and weaknesses and are therefore applicable in different use scenarios. However, training on real data has proven difficult and requires some effort to reconstruct the original historical font from cut-out glyphs. This was also achieved by the team in Poznan (Poland) in paper [36] with the *Franken+*[37] tool. The authors reported that this tool managed to achieve a character accuracy of about 86% on the ECCO document collection [38].

A completely different approach was adopted with the OCR engine Ocular [39], which can convert printed text to electronic text in a completely unsupervised manner (i.e., no need for a ground *truth dataset*). This may be a viable alternative for training individual models with reduced manual effort, but it appears to be very resource-intensive and slow (transcribing 30 lines of text takes 2.4 minutes). Its results are better than Tesseract and ABBYY FineReader (without training), but it remains to be demonstrated that they can consistently achieve character recognition performance greater than 90%.

In paper [40], it is shown how the recognition accuracy for historical Polish texts can be significantly improved by training the OCR engine used. The report shows improvements from 45 to 80% in character recognition accuracy and 15–55% in word recognition accuracy for recognizing Gothic documents after training ABBYY FineReader on just a few pages. To leverage OCR training, systems like ABBYY FineReader have basic built-in facilities to add at least a few new symbols to an existing language [41].

Tesseract and other open-source programs, on the other hand, offer more extensive training possibilities. There are various works on how specific OCR training issues have been addressed.

One example [42] demonstrates how Tesseract was trained to recognize Ancient Greek. The process described mainly relies on scripting and some manual interventions. The authors provide some general recommendations, but no systematic approach regarding the choice of parameters and settings is described, nor are any statistical justifications presented.

Another example can be found in [43], where the authors train the Tesseract engine to recognize *Odia* – an Indian script. Here, the training set is first generated artificially and then introduced into Tesseract's standard command line tools. Although this might work for languages (character sets) that are fundamentally known, it wouldn't be a viable option if unknown characters and symbols are expected to appear (which is quite common in historical documents).

In the paper [44], the authors describe an efficient approach for training OCR engines using the *Aletheia* document analysis system [45]. The development of the Aletheia system initially began in 2001 with the goal of creating a system to produce ground truth datasets for page structure and text block recognition. Since then, Aletheia has evolved into a comprehensive document image analysis system, incorporating support for multiple file formats, image processing/enhancement and geometric correction methods, integrated OCR, functionalities for entering and manipulating text blocks, and more. The general architecture of the Aletheia system has a modular design that allows for the integration of various OCR engines. The Tesseract engine is available by default in Aletheia. Communication between modules is based on a command-line interface, and data communication is done through PAGE XML (with compatibility also with other formats like ALTO and FineReader XML) [46]. All the necessary components for training OCR models are integrated into Aletheia: preparation of training data, the training processes of the OCR model itself, text recognition, and evaluation of the trained model. Such a training and evaluation system, guided through a user-friendly graphical interface, allows for iterative incremental training to achieve the best results. Besides the detailed description of the proposed OCR engine training system, the paper [47] also reports several experiments conducted on various datasets to investigate the ideal training conditions in terms of the size and quality of a training set. The authors validated the

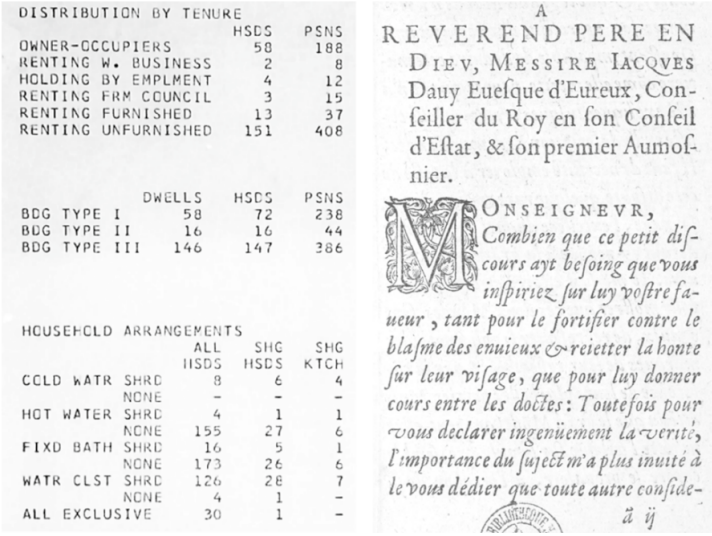
training process's effectiveness using two very different datasets, each representing a realistic use-case scenario where OCR engine training can make a difference: a dataset from the 1961 Census for England and Wales (like more "modern" monospaced fonts) and a book from the French National Library printed in 1603; the data being collected and grounded for the IMPACT project [48].

In Fig. 3.1, p. 32, the authors present two examples, and Table 3.1, p. 31 displays the characteristics and sizes of the training and testing sets. All experiments were conducted by creating an initial set of training examples (glyphs along with ground truth), and the results were evaluated using a text-based metric. The training process was carried out through Aletheia.

Dataset	Characters/font	Training Set	Testing Set
1961 Census	40 character classes, uppercase Latin letters, digits, and punctuation marks; a single font	2150 glyphs	2115 glyphs
French National Library (1603)	74 character classes; Latin with French characters and ligatures, two fonts	773 + 1321 glyphs	2040 glyphs

Table 3.1: The datasets for model evaluation in [47]

Special characters and ligatures were not transliterated because were introduced in *diplomatic transcription* mode. Diplomatic transcription[49] tries to represent document exactly as it appears, without changes such as expanding abbreviations, transliterating proper names, etc. The strategies considered by the authors for selecting training examples involved: removing partially erased/deformed glyphs and eliminating glyphs with a similar appearance. The impact of each strategy was tested by gradually eliminating more and more training examples from the dataset. To measure the OCR results' quality, a tool called *TextEval* [50] was used. Among other measures, it uses an implementation of the University of Nevada's measure [51], which is based on the string's edit distance. The quality of evaluation is reported in percentages, where 100% indicates that the text was recognized perfectly.



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
COLD 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-
ā ij

Figure 3.1: Examples of pages from the evaluation dataset in paper [47]

To measure the impact of the trained models, OCR results with and without training were compared. *Without training*, in this context, means using the default linguistic information files provided by OCR engines. For the sake of a complete comparison, the authors also evaluated the commercial system ABBYY FineReader Engine 11. It should be noted that the *Census* dataset includes only uppercase Latin characters (as well as digits and punctuation marks).

Table 3.2, p. 32 presents the evaluation results made by the authors.

OCR Engines and Models	OCR Accuracy by Dataset (%)	
	1961 Census	French National Library (1603)
FineReader (without training)	88.40	88.40
Tesseract (without training)	90.08	84.93
Tesseract (with training)	95.40	90.14

Table 3.2: OCR accuracy with and without training in [47]

The trained Tesseract engine outperforms all other configurations, and

the authors managed to investigate the accuracy for the FineReader engine *with training*. For both testing sets, there is an approximate 5% increase in performance compared to the Tesseract OCR results without training.

If we were to discuss free OCR tools, then the most popular ones are *OCRopus*[52], *Ocropy*, *Kraken*[53], *Tesseract*[35] and *Calamari*[54]. Ocropy and Kraken train a neural network with an *LSTM* architecture. Long short-term memory (LSTM) is an artificial recurrent neural network that can process not only individual data points (such as images) but also entire sequences of data (such as audio or video). Ocropy and Kraken architecture is constructed with a single layer of neurons, while the newer versions of Tesseract and Calamari train OCR models with multilayer networks of types of LSTM and *CNN*. A Convolutional Neural Network (CNN), also known as ConvNet, is a category of artificial neural networks primarily used for processing and identifying images[55].

In the paper [56], various OCR methods with Ocropy are applied to historical printed documents with a Latin font, resulting in good accuracy. The authors of [57] present the architecture of Ocropy and explain the various steps of an OCR process. Springmann and Lüdeling in their work [30] (discussed above) use Ocropy to recognize printed documents between 1487 and 1870 and report a character-level performance of over 90%.

In the paper [58], the authors introduce for the first time the software called Calamari (also known as Calamari OCR) - a set of tools for training and recognizing text lines from images. It was created as an improved version compared to Ocropy. Calamari supports a user-defined multilayer CNN-LSTM neural network architecture, which has been found to increase model accuracy. They use Tensorflow as the backend, which appears to enhance computational performance compared to Ocropy, especially during training and recognition on a GPU. The Calamari tool can serve as a replacement for Ocropy and offers other important features as well. For example, models can be trained and text can be recognized on a GPU, improving performance. The implementation of additional features such as The implementation of additional features such as:

early stopping - in machine learning, early stopping is a form of regularization used to prevent overfitting of a model when it is learned using an

iterative method such as *gradient descent*;

cross-validation[59–61] or out-of-sample testing are techniques for validating a machine learning model to assess how well the trained model's results will generalize to an independent dataset;

pretraining has led to lower error rates [62] and works as follows: let m is a machine learning model and A is a dataset, on which you train m, therefore - before starting the training of model m on the new dataset B, m is (pre)trained on A.

In the work [63], Calamari's performance is tested compared to Ocropy on historical documents, demonstrating that a combination of a convolutional neural network and an LSTM network performs better than a single LSTM layer (used in Ocropy). It was found that to achieve an error rate below 2% for recognizing a book, a pretrained model requires training examples with ground truth from 60 lines of text. The authors verify both training and recognition speeds. Training the neural network in Calamari is faster than training Ocropy when using multiple CPU cores (more than 4). However, training a model on a GPU is 4 times faster. In the prediction phase, Calamari is 3 times faster than Ocropy even with a single CPU core and approximately 30 times faster on a GPU.

In the work [62], the authors use the Calamari tool to recognize a corpus of historical newspapers published in Finland between 1771 and 1929 [64]. It's worth noting that this corpus was previously recognized with ABBYY FineReader 11 and has a character-level accuracy ranging from 87% to 92%, which is quite high for qualitative linguistic analysis of the corpus. However, the authors considered the need to re-recognize the entire corpus of documents using the advantages of the Calamari tool. This corpus contains highly diverse data written in a non-standard language [62]. The newspapers in Finland from the 18th century to the early 20th century were printed in two basic languages of Finland (Finnish and Swedish) using two different font families: *Gothic* (Blackletter) and *Antiqua*. It's worth noting that the data is not evenly distributed. In older documents, there is more material printed in Swedish with Gothic fonts, while modern documents are mostly printed in Finnish with Antiqua fonts. However, there are periods when both languages and font families were widely used, sometimes even on the

same newspaper pages. The standardization of the Finnish literary language began in the 19th century [65]; therefore, a significant portion of the corpus contains spellings from different Finnish dialects [45]. The size of linguistic corpus is usually measured by *tokens*. A token is an instance of a character sequence in a specific document, grouped together as a semantic unit useful for processing. Tokens are often loosely referred to as *terms* or *words*. The task of segmenting a text into tokens, while discarding certain characters such as punctuation marks, is called *tokenization*. The corpus from work [62] is very large, with nearly 5 billion tokens, so the authors also try to find an efficient method for the repeated recognition of the corpus. The OCR approach proposed by the authors includes image preprocessing, where the basic process is the conversion to black and white; image segmentation into text lines (Fig. 3.2, p. 35) training examples consist of text lines and character sequences themselves - it's very important here to note this difference in approach compared to OCR engines that train with glyph-level training examples, such as Tesseract 3 or ABBYY FineReader.

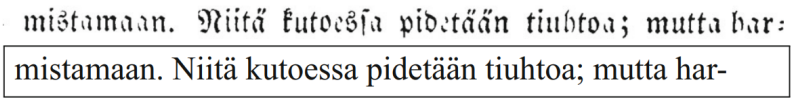


Figure 3.2: An example of training with a line of text from an image and the matched ground truth sequence of characters[62]

It is agreed that correctly segmenting each glyph is difficult, leading to many segmentation errors, and creating glyph-level training templates for different fonts, especially for historical documents, is very time-consuming. The ability to present whole lines of text in LSTM neural networks, line-level segmentation asserting itself as the latest technology [62]. Next, in the recognition process, the dataset is prepared, and a recognition model is trained, resulting in text that can be corrected using linguistic post-processing methods. The result is usually in XML files. For the preparation of the training and test set, already available data collections are used [66], which consist of approximately 9500 lines of Finnish text and 6500 lines of Swedish text (418 from each set are used for testing only), as well as separate datasets created by the authors. Both datasets were randomly extracted from the

corpus of historical newspapers and magazines [64]. The Finnish dataset is extracted from the period 1820-1939, while the Swedish dataset covers the years 1771 to 1874. In addition to existing datasets, the authors added 5000 lines from Swedish documents and 4000 Finnish lines from the same corpus, manually transcribing them. To obtain training examples from the corpus, the segmentation and OCR information of ABBYY FineReader, stored in files *METS-ALTO*. The METS standard is a flexible schema for describing a complex digital object (such as a digitized newspaper issue). METS describes the structure of the object but does not encode the actual textual content of the object. The ALTO standard fills this gap by encoding the textual content of a digitized page in detail, including styles and layouts. In addition to encoding the digitized text itself, ALTO encodes the spatial coordinates of each column, line, and word as they appear on the page. METS and ALTO are XML standards maintained by the U.S. Library of Congress[67]. The authors obtained approximately 11,500 lines of Swedish text and 13,500 lines of Finnish text after preparing the dataset, both consisting of a similar percentage of Gothic and Antiqua font lines. The Finnish and Swedish test data sets, in particular, each contain 418 lines of text. To test the OCR model's results, the authors use cross-validation on the dataset divided into 5 equally sized parts. The following measurements are conducted for result evaluation: Character Error Rate (CER) and Word Error Rate (WER). The Character Error Rate is the percentage of incorrectly recognized characters out of the total number of recognized characters. Similarly, the Word Error Rate is the number of incorrectly recognized words divided by the sum of correctly and incorrectly recognized words. To obtain the number of errors, the authors first aligned the ground truth with the recognized text lines at the character level (for both CER and WER) and calculated the *Levenshtein* distance [68] between them. In the recognition phase, the authors of the work [62] use the prediction tool from Calamari. From this perspective, the main advantage of this tool is the ability to run on a GPU, which makes the recognition phase very fast. Another feature of the tool is the capacity to recognize an image with multiple models at the same time and then to choose the optimal result using a voting mechanism. The *voting mechanism* is a *meta-machine* learning model that combines predictions from several other models.

This is a technique that can be used to improve the model's performance, ideally achieving better performance than any single model. Each model predicts multiple candidates along with their associated probabilities, and then the voting mechanism decides which candidate wins. The authors of the study [64] demonstrate that voting always reduces errors. The disadvantage of this method is the need for recognition using multiple models, which slows down the recognition process. To evaluate the performance of the recognition models, the authors conducted experiments with mixed models (models trained on both Finnish and Swedish data); monolingual models (models trained on either Finnish or Swedish data); application of the voting mechanism on combinations of models; and post-correction. The mixed models achieved an average error of 2.6% CER and 10% WER. Specifically for Swedish, they obtained 3.8% CER and 13% WER, while for Finnish, they produced 1.7% CER and 8% WER. Applying the voting mechanism on models that operated with the Swedish test set yielded 2.8% CER and 11% WER, and the Finnish test set generated the best results from 5 equal mixed models with 1.9% CER and 8% WER. Finally, the authors performed post-processing (error correction after recognition) on the new OCR results. The results showed a significant increase in accuracy, resulting in 1.7% CER on the Finnish test set and 2.7% CER on the Swedish test set. The authors' greatest achievement is the successful formation of a mixed model for the entire corpus and finding a configuration of the voting mechanism that further improves the results.

Next, we will analyze some **platforms** or **frameworks** for digitizing and processing historical documents developed in major projects related to the digitization of cultural-historical heritage. Certainly, the process of digitizing and processing historical documents would be more efficient and streamlined if all the necessary tools were available in one place and integrated into a single software application. Such applications could be referred to as *digitization platforms* or *frameworks*.

HDPA. In the paper [69], a complex and flexible web framework named *Historical Document Processing and Analysis Framework (HDPA)* is described for managing and analyzing historical documents, with a primary focus on OCR. The HDPA framework is available for free [70]. The framework contains eight modules to facilitate three main tasks: image preprocessing and

segmentation, creating a dataset for training the OCR model, and the recognition itself. This framework is available for free for research purposes. The authors demonstrate that this system is efficient and can save human labor in the process of preparing datasets for OCR. The web application is written in *Django*[71]. Django is (free and open source) high-level web framework, written in Python, that encourages rapid development and provides a clean, pragmatic design. The Django implementation of HDPA allows developers and researchers to develop individual Python modules. Modules can run separately, and Django acts as a hub connecting the user interface with the desired module outcomes. The proposed HDPA framework contains units or functional groups that perform three main tasks. The first functional group deals with image preprocessing and page segmentation. The second functional group provides tools for creating datasets (*ground truth*) for training the OCR model. The third functional group encompasses the OCR engine. A post-processing module is not integrated by default in the HDPA framework, but the authors ensure easy integration of new modules. This way, users can easily customize the HDPA system for their specific needs. The framework currently offers users two basic preprocessing modules, namely image binarization and rotation. The training dataset creation module is useful when we want to train a new OCR model. It provides tools for creating a set of images with a single letter/glyph, cut out directly from images uploaded by the user. Another way to create synthetic datasets is by using a text image generation tool. Here an OCR model is trained in two stages: training on large synthetic data, which helps in learning the shapes of glyphs; and training on a small amount of images with text lines cut out from real pages, which ensures that the model can learn some specific aspects of real data, which cannot be generated in synthetic lines. To allow the creation of ground truth datasets for real data, the authors offer a tool for annotating text lines. The OCR system proposed by the authors is based on machine learning and uses CNN networks for feature extraction and a bidirectional LSTM recurrent neural network for sequential recognition of text lines. In Fig. 3.3, p. 39, the authors present the architecture of the HDPA framework.

The architecture is modular, consisting of eight modules (*M1–M8*) encapsulated in three functional units (*U1–U3*)[69].

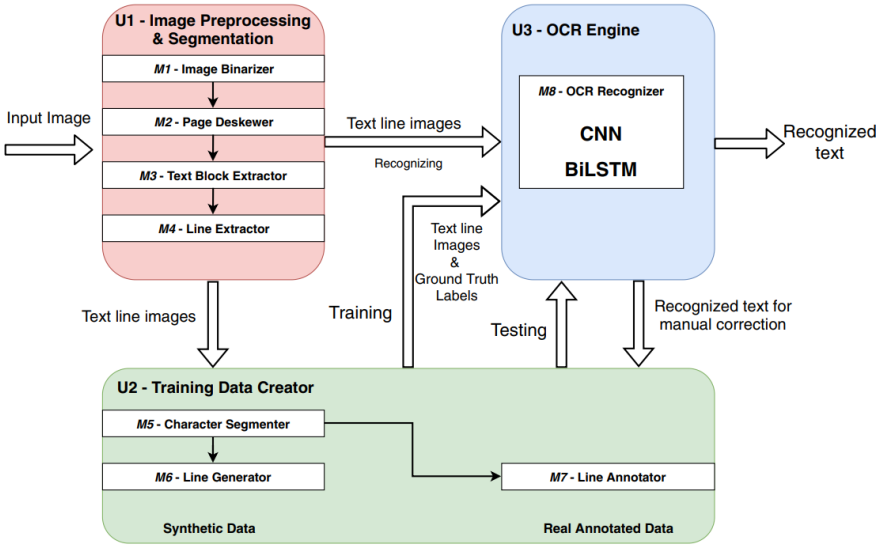


Figure 3.3: The architecture of the HDPA framework [69]

The first functional unit *U1* deals with image preprocessing and segmentation. Preprocessing includes important image transformations and corrections, which are necessary for successful image analysis and segmentation. The purpose of unit *U1* is to prepare text lines from the image for training the OCR engine. Module *M1* in *U1* performs image binarization. The binarization module in [69] uses the adaptive thresholding method proposed in the paper [72].

The next module,*M2*, deals with rotating the image to the left/right so that the rows of text are horizontally aligned. This way, line-level segmentation will yield better results.

Next, module *M3* detects and extracts text blocks from the straightened pages in module *M2*. In this module, the authors implemented the convolutional U-Net network [69, 73] and trained it on two different training datasets: the first training dataset was from the *Europeana* project [74], and the second dataset was created from historical documents from the *Porta fontium* project [75] dealing with archives from the Czech-Bavarian border area. A result of

module *M3* is shown in Fig. 3.4, p. 40, where bounding boxes are identified and drawn on the original image.

Figure 3.4: Image processing with module *M3* from HDPA [69]

The last module in functional unit *U1* is *M4*, which segments and extracts text lines. To implement this functionality, the authors use ARU-Net [69, 76], a multi-layer neural network designed to detect baseline lines (the line on which the letters sit) in manuscripts. This neural network can detect lines on pages with variable font size, but for text lines, the authors determine the font size, they use an algorithm based on a projection profile of a region above the detected baseline. A visualization of the text line segmentation process, applied to one of our examples, is presented in Fig. 3.5, p. 41).

The next three modules (*M5*, *M6*, *M7*) are part of the functional unit *U2*. This functional unit is used for creating/generating training data for the OCR model. Unit *U3* contains module *M8*, which is the OCR engine itself. The engine uses an approach based on processing text lines, which recognizes the images of extracted text lines and generates the predicted character sequence. Module *M8* handles both the training phase and the recognition phase. The developers of the HDPA framework have included in module *M8* pre-trained models on a synthetic dataset containing 25,000 images with text lines.

The texts used for generating synthetic images are based on old German documents to ensure that the language corresponds to that used in the

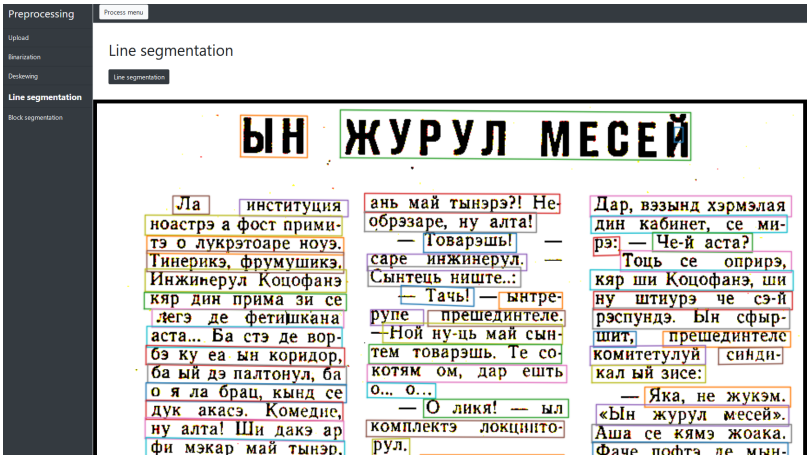


Figure 3.5: Image processing with module *M4* from HDPA

documents processed by the authors. They also used the annotated *Porta fontium* dataset for training. The OCR engine proposed in module *M8* uses a combination of a convolutional and recurrent neural network. CNN is used for feature extraction, while LSTM is used for the recognition itself. The architecture is a simplified version of the network proposed in paper [76]. For evaluating OCR results, the authors use average accuracy/precision at the text line level, WER and CER based on the results of cross-validation on 10 test pages divided into 5 equal parts. Recall that WER is the word error rate, and CER is the character error rate. In the case of paper [69], the average precision indicates how many images with text lines out of all processed were correctly recognized, with the authors achieving a result of 0.488. The results for WER and CER are 0.11 and 0.024, respectively.

The authors mention that future development of the framework will focus on building an extension that will allow the application of natural language processing methods on the transcribed data, which will include such tools as named entity recognition, classification, and intelligent full-text search in documents.

Aletheia. Continuing to discuss digitization, processing, and analysis platforms for historical documents, we will repeatedly mention in this context

the Aletheia platform[45]. In Aletheia, a particular focus of the authors is on the analysis of the document page's layout and page segmentation. Document segmentation or document layout analysis is the process of identifying and classifying regions of interest in a scanned image of a text document. A reading system requires the delimitation of text areas (blocks) from non-textual areas and the correct order of reading them [77]. Detecting and labeling different blocks, such as text blocks, illustration blocks, mathematical symbols, and tables embedded in a document is called geometric layout analysis [78]. Text blocks play different logical roles within the document (titles, subtitles, footnotes, etc.), and this type of semantic labeling is the scope of logical layout analysis. Aletheia can automatically detect objects on four levels: regions of interest (text, tables, formulas, musical notes, etc.), text lines, words, and glyphs. The contours of objects can be adjusted by the user. The creation of ground truth is another feature of the Aletheia platform. Ground truths are stored in the PAGE XML format [47].

Transkribus. Another digitization platform is *Transkribus* [79] - a complex tool developed within the *READ project* [80] at the University of Innsbruck, which deals with the recognition, transcription, and searching of historical documents. It offers a range of tools for the automatic processing of historical documents, such as handwriting recognition, page layout analysis, document understanding, writer identification, or optical character recognition (OCR). For OCR, Transkribus uses the ABBYY Finereader engine. Transkribus does not support the generation of any type of synthetic data, a feature well implemented in the HDPA framework [69].

OCR-D. Recently in Germany, the *OCR-D* project [81] emerged, featuring 8 special modules focused on various stages of OCR. The project's architecture with the workflows is displayed in Fig. 3.6, p. 43.

Along with this project comes the *OCR4all* platform [82], an open-source tool providing a semi-automatic OCR workflow for historical documents.

Besides discussed above *general historical documents digitization platforms*, there are many tools, which concern **specific problems**, for example, the *generating artificial OCR datasets* for Russian [83], Arabic [84], and Romanian [85] languages.

Considering this, we can conclude that the value of digitization platforms

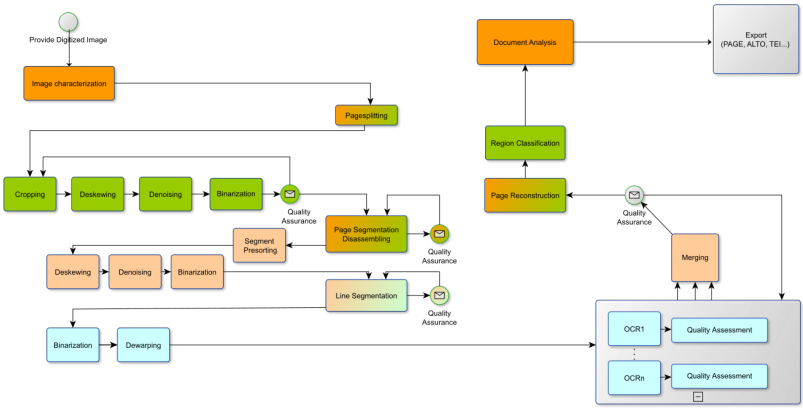


Figure 3.6: A segment of the project's structure OCR-D

with *full functionality* is much greater.

3.2 OCR tools for Romanian historical printed texts

An important project in this field is ***DeLORo*** project PN-III-P2-2.1-PED-2019-3952, no. 400PED: *Deep Learning for Old Romanian* [86]. The objectives of this project include the development of technology capable of deciphering documents written in the Romanian language with Cyrillic characters and transliterating them into Latin characters, thus laying the foundation for opportunities to study and preserve cultural heritage [87, 88].

Within the project, the authors managed to develop an online tool for annotating images from Romanian Cyrillic documents. Moreover, the groundwork was laid for the creation of a valuable resource, namely the Old Romanian Cyrillic Corpus (*ROCC*) [88], which includes a collection of scanned historical documents annotated with transcribed text [89]. The corpus consists of 367 scanned document pages, with a total of 6418 annotated lines of text.

Details regarding the resources and technology developed in the DeLORo project are described in Cristea et al. [88]. This paper proposes solutions for creating the dataset in the training process of neural networks and their

use in training the actual neural networks for tasks such as segmentation and OCR. In this endeavor, the authors construct the ROCC corpus. The collection of documents in ROCC covers the 16th to 19th centuries and deals with various levels of quality in historical documents.

The process of collecting documents for this corpus is iterative and includes original images of pages in Cyrillic script, and manual annotations *regarding the segmentation of columns, text rows, words, and characters*(Fig. 3.7, p. 44).

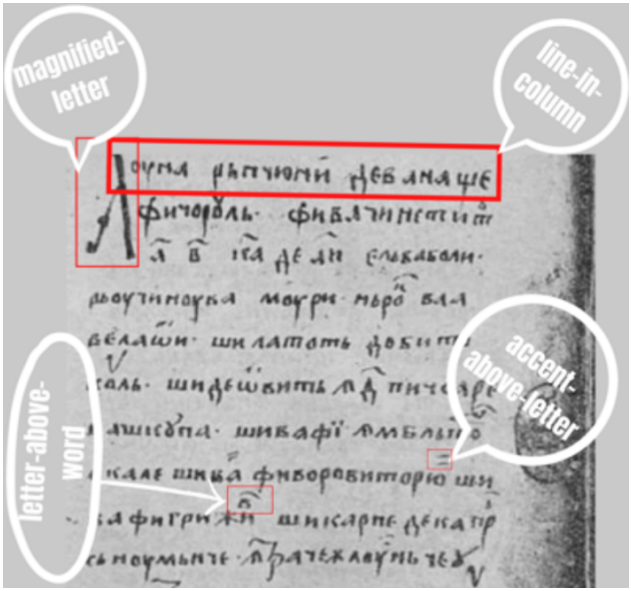


Figure 3.7: Annotated objects on a scanned page[88]

Manual annotation involves two components: visual segments in *columns, and rows, interlinear or marginal writing, words, characters; and their transcriptions in Latin script.* When developing the resources needed for training neural networks, the authors focus on two types of them: a language model containing as many word forms as possible ever written in the Romanian language in Cyrillic script and a large collection of identified and transliterated graphemes. Collecting the first type of resources is very challenging for

several reasons. One reason is that in many historical periods, there were no language usage norms, so there is a great diversity of written word forms over time and space. Inflection patterns for Old Romanian have not been identified, which would have allowed for the automatic generation of the old Romanian lexicon; proper nouns are almost impossible to inventory in their entirety. This type of resource acts as a vocabulary, and the authors propose the application of automatic augmentation methods to it. The second type of resource in the ROCC corpus is used for training neural networks on different document qualities and different types of writing. These resources should assist in the classification task, where visual objects that appear on a scanned page are identified and labeled.

Undoubtedly, large training datasets are required when training neural networks. In the OCR stage, the authors train neural networks with pairs of Romanian Cyrillic graphic symbols and their corresponding Latin transcriptions. In addition to the annotations made through the *OOCIAT* interface, the authors also use alternative resources from *Monumenta Linguae Dacoromanorum (MLD)* [90] and the *UAIC-RoDia Treebank* corpus [91] – a collection of syntactic sentence trees in Old Romanian. The volume of the dataset is still insufficient. One method to augment the proposed dataset is the iterative use of what has been manually annotated for the OCR stage, followed by manual correction of the OCR results. The authors believe that this process can accelerate manual annotation because, for the next set of new pages, the accuracy of the OCR modules would be higher, and accordingly, the manual correction effort should decrease. Another proposed method is to use a manually transcribed document and align the images of the pages with the corresponding text, line by line.

Considering that the results consist of sequences of characters delimited only by line breaks, and given that in old Cyrillic printings, words are rarely separated by distinguishable spaces, the authors propose methods for word segmentation. Assuming that each character in the original image of a text row is recognized, the problem of segmenting the sequence of Romanian Latin letters into words remains. To address this, a *sequence-to-sequence*[92] approach is used, which takes a sequence of letters as input and, for each individual letter, recognizes one of the 4 classes:*beginning of a word, end of*

a word, middle of a word, and single-character word. The authors do not use any context to disambiguate ambiguous Cyrillic letters (examples include: ѫ ѧ: ŭa/ea; ѳ ҧ: th, ft). Another action implemented by the authors is lexical clustering to group old word forms belonging to the same lemma and part of speech. To achieve this, the authors intend to apply *string kernels* [93] and *spectral clustering* [94]. In the first step, the inflected forms of the same lemma found in a collection of old Romanian Cyrillic documents will be labeled as belonging to the same class. In the second step, the detected clusters should be aligned with the dictionary entries of a modern Romanian language dictionary. In their ongoing work, the authors plan to integrate the developed tools into a freely accessible platform for researchers.

3.3 OCR post-processing methods

OCR post-processing or OCR post-correction is a task that is either manual, semi-automatic, or automatic for checking and editing the text recognized by an OCR engine. Integrated post-processing in an OCR system adds significant value to the obtained result, making the OCR system more robust and valuable in terms of mass digitization of historical documents.

There are many different approaches to OCR post-processing.

Some of the basic methods view post-processing as a task of correcting the text's spelling [95–98].

In the paper [62], analyzed above, the authors, basing on he work [99], use a method called *sequence-to-sequence* or *seq2seq*[92]. *seq2seq* is a family of machine learning approaches used in natural language processing. This approach is applied to problems such as machine translation, text paraphrasing, text alignment, and summarization. This method can use a vocabulary/dictionary of words or lexicon to identify OCR errors, but it can also work without a dictionary. The method corrects input tokens and creates an error model, which is a set of context-dependent rules annotated with weights, implemented as a weighted finite-state automaton. After the error model is created, a dictionary lookup is used to validate or eliminate suggestions generated by this model. For training OCR post-correction models, the authors took a text recognized using a voting mechanism with 5 mixed models [62]. The

best results after post-processing the recognized test datasets were achieved with dictionary-less post-correction with 2.7% CER for Swedish, 1.7% CER for Finnish. The dictionary-less post-correction managed to improve all results, and post-processing with a dictionary did not produce any change in the result [62].

Most OCR post-processing methods have at least two steps:

1. **Generation of candidate** suggests the replacing incorrect tokens;

2. **Decision-making** accepts (or does not) the proposed corrections from the *first step*.

Other approaches would involve three consecutive steps:

1. **Vocabulary expansion** where the initial word dictionary is extended with frequent words from the recognized texts;

2. **Candidate ranking** where for all presumed errors, corrections are calculated using an analytical rule-based approach, and the proposed candidates for correction are sorted by the most likely candidate;

3. **Decision making**: finally, it is decided whether the presumed error is replaced with the correction suggestion or left unchanged.

Machine learning models are used to estimate the risk of incorrectly replacing a correct word. Therefore, the decision step helps avoid erroneous correction steps. In this context, the authors of the paper [100] carried out OCR post-processing of newspapers printed in German between 1910 and 1920. They use a dictionary and an external corpus along with the Levenshtein distance and *n-gram* frequencies to vote for candidates and find the winner (the candidate with the most votes). N-grams are sequences of elements as they appear in texts. These elements can be words, characters, or any other units as they occur one after another. The "N" or "n" in the term "n-grams" corresponds to the number of elements in the sequence [101]. Their model performs well when the correct result does not have a Levenshtein distance from the candidate token greater than 2.

Another example in this direction is described in [102], where historical Arabic documents with OCR results below 70% word-level accuracy are post-corrected to achieve over 90% word-level accuracy. They use a word dictionary to search for and identify incorrect words in the recognized documents, for which they create correction candidates using a regression

model. In the second stage, the candidate is again selected using a regression model, but this time based on word features obtained from a language model, constructed from a large text dataset.

Manual OCR post-correction can yield high-quality results, often better than fully automated ones, but it requires time and effort, sometimes even specialists, when dealing with historical documents printed in old alphabets no longer in use [103]. People performing manual correction need to have a good understanding of the language of the document and be trained in how to use post-correction tools. There are semi-automated approaches that make manual correction easy and efficient. In the approach presented in [104], an interactive interface is proposed for correcting the results of handwritten text recognition. The user's corrections are taken into account in real-time and are subsequently used to provide better correction suggestions.

A tool for semi-automated OCR text correction is *PoCoTo* [28]. The tool was initially developed as a desktop application and later as a Web application. PoCoTo uses a mechanism that calculates with a certain probability errored words in an OCR document, using a combination of statistical and analytical methods. The graphical user interface presents the user with page snippets along with OCR text that contains potential errors and offers possible corrections. Thus, it is easier for users to manually correct the errors. In 2019, an improved and fully automated version of PoCoTo, named *A-PoCoTo* [105], was published. The OCR approach of this tool has a machine learning model to classify candidates that require correction. At its input, A-PoCoTo receives a set of $\boldsymbol{n \geq 1}$ parallel OCR results for a historical document. To obtain $n > 1$ parallel OCR results for a document, multiple OCR engines can be used. After the preliminary alignment of all available OCR results, the OCR post-processing for a document begins with the expansion of the word dictionary. Input documents often contain special names (personal names, geographical names, etc.) and other expressions. Any word not found in the dictionary is considered a candidate for expanding the dictionary. Intuitively, if there is evidence that a word in the recognized text is not a recognition error, then it makes sense for the user to include it in the vocabulary. This implies that this word can serve as a candidate for subsequent corrections.

In the paper [106], an interesting method is presented that is based on the

observation that OCR errors for the same word, due to semantic similarities, are grouped in the vector space. The authors obtained clusters of OCR errors and synonyms of words, using a *Word2Vec*[107] family of model architectures and optimizations. Then, by verifying correct words using a dictionary, the groups of incorrect and correct words are identified using the *Levenshtein* distance. Utilizing this parallel dataset, a neural machine translation model is trained, performing a translation of incorrect words into correct words.

A similar approach to that in [106] is presented in [108], where the authors propose an OCR post-processing tool that also groups all similar words in the text. However, instead of grouping them semantically, they group them based on character distances in Euclidean space. They use external data such as lexicons and morphological rules to suggest correction candidates. Ultimately, the correction is performed by selecting the candidate with the highest score.

3.4 An overview of strategies for integrated recognition platforms

The process of addressing the challenge of digitizing heterogeneous content involves several key stages: preprocessing, layout classification/analysis, assembly, and digital preservation.

Our work delves into the preprocessing stage in detail [109]. Over recent years, deep learning-based image preprocessing has become a standard tool in numerous optical character recognition systems, both commercial and open-source. In the classification/label analysis stage, image processing of heterogeneous documents is segmented based on content type. Digitizing heterogeneous content for general cases, where any content type is allowed, is resource-intensive. Thus, we separate the classification step from the workflow. This one-time event typically occurs during the digitization of document archives, with subsequent additions through an interface specifying the type of heterogeneous content. The unique nature of the problem makes it well-suited for processing on online platforms. Major commercial systems like Canon IRIS [110] and Kofax Layout Classifier [111] are part of the growing set of such platforms.

Apart from commercial systems, individual projects, particularly research initiatives, are developing their platforms, mainly grounded in deep learning [112]. For instance, the SCyDia OCR application [113] focuses on the Serbian Cyrillic script with diacritics, used in dictionaries and linguistic publications. SCyDia, implemented in Python, integrates Tesseract [35] and includes neural networks for letter and diacritic recognition.

While some projects deal with general classification cases, most focus on specific types of heterogeneous content. General case classification projects typically leverage deep learning [114, 115]. An intriguing study [116] addresses the processing of distorted images of heterogeneous documents caused by scanning. Another unique approach is proposed in [117], showcasing a hybrid cross-modal methodology learning from both text corpus and structural image information. A non-Deep Learning approach is demonstrated in [118], suggesting a novel document classification technique based on formal concept analysis. Legal heterogeneous document images [119] represent a specific case where classification, rather than layout analysis, is crucial. Legal documents often have strict layout patterns, requiring categorization for digital conversion.

Despite being fully developed, processing text-type layout elements also finds interesting applications, such as specific text area searches on ID cards [120].

Upon reviewing tools and platforms for digitizing historical documents, it's evident that numerous methods, tools, resources, and platforms, both existing and in development, offer pre-processing, recognition, post-processing, and translation of historical documents. Additionally, novel recognition methods, like training OCR patterns on images with suggested text strings [44], enhance OCR mechanisms' learning rates and expedite mass digitization. Post-recognition OCR tools offer high-quality text for further language processing, web placement, preservation, and metadata enhancement in digital libraries.

A noteworthy contribution is the emergence of digitization platforms streamlining tasks for successful historical document processing. The DeLORo project [86, 89] showcases significant efforts in deciphering old Romanian documents, presenting promising alternative recognition solutions.

Chapter 4

Recognition of the oldest Romanian texts

4.1 Introduction

In this chapter, we will elucidate our technology designed for processing historical texts, with a specific emphasis on the oldest Romanian Cyrillic printings from the 17th and the majority of the 18th centuries. Throughout this period, the Romanian Cyrillic alphabet underwent significant evolution. We will specifically discuss the Romanian Cyrillic (RC) variant consisting of 47 characters, occasionally referencing its simplified counterpart (SRC) as outlined in Section 1.1 above.

In the 17th century printings, the action of introducing the Romanian language into the Church was intensified, which meant replacing the Slavonic language as the liturgical language with Romanian. This was an extremely important achievement, conclusively accomplished in the early 18th century. Despite the fact that Romanian is part of the group based on the Latin language, the script in which the texts were printed was Cyrillic (Slavonic). Considering that most printing presses were located in sacred places (churches, monasteries, etc.), the majority of printed documents and books also had religious themes.

The main source of digitized Romanian documents from the 17th cen-

tury is the Digital Library of Romania. The archive of this library includes approximately 490 scanned old Romanian books, out of which 80 books are from the 17th century.

Our technology for processing historical Romanian Cyrillic documents includes OCR and transliteration technologies. The OCR technology includes: image preprocessing; document layout analysis and segmentation; and font classification. The transliteration technology will be described in chapter

4.2 OCR process using ABBYY FineReader 12

The analysis in the previous chapter supports the use of ABBYY FineReader Professional (AFR) for recognizing Romanian Cyrillic characters. Initial tests and adaptations were conducted on version FR 12, with subsequent preference for newer versions like FineReader 14 and FineReader 15. Both AFR 12 and the later versions are not inherently geared towards processing old Romanian texts. Therefore, our efforts focused on expanding the capabilities of this product to address these challenges. To work with documents from the specified period, we developed new components, especially alphabets and dictionaries, and trained the software on additional datasets for high-quality results. These actions culminate in the creation of one or more models tailored for recognizing texts from a specific historical period.

AFR is an OCR software equipped with the necessary functions for preprocessing and recognizing documents, the use of which significantly contributes to increasing the productivity of these operations. It provides high-performance tools that are easy to use for accessing information contained in printed documents and PDF files.

In Fig. 4.1, p. 53 we present a document from the 17th century processed with AFR 12.

The processing includes applying the optimal resolution (800-1200 dpi in this case), manually removing wear and tear spots from the image, training the OCR model, recognizing all pages of the document, and manually correcting the OCR results.

We would like to mention that some image processing operations, such as binarization and straightening of lines, were performed with another

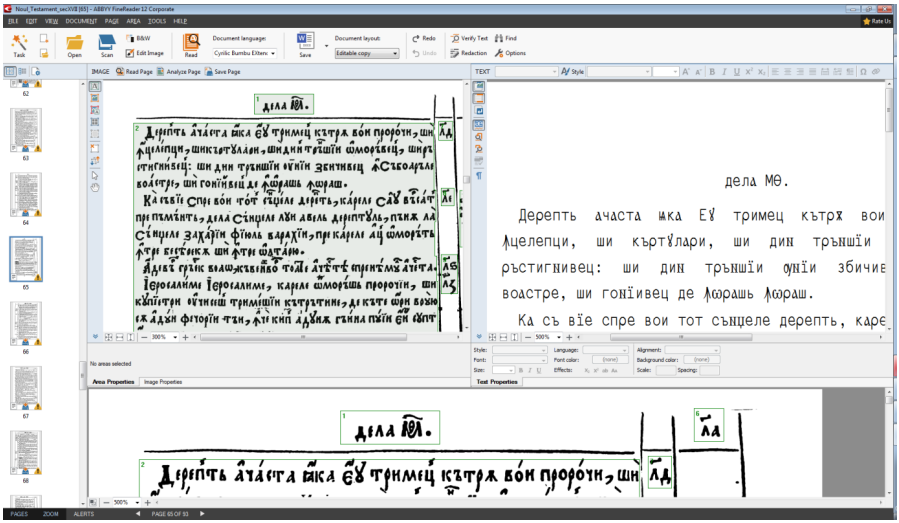


Figure 4.1: Main window of AFR 12 graphical user interface

software tool, namely Scan Tailor. We will discuss this tool in the next section.

In case when a document printed in a language not supported by FR, a new language named "user's language" has to be added. The OCR process of such case contains several additional stages.

Below is a list of the most important stages of OCR of a document printed in the "user's language".

1. ***Image Preprocessing.*** This stage usually involves editing the image, removing irrelevant elements or noisy content from the image, straightening the lines, and adjusting the resolution to an optimal level. In our case, we can perform these actions with existing software, but finding a specific way to apply them to old texts.

2. ***Language and Alphabet Preparation and Creation.*** At this stage, all constituent characters (letters, punctuation marks, ligatures - referring to the combination of multiple letters into a single graphic symbol) of the relevant language are prepared. If the alphabet contains characters from multiple languages, the base languages are first selected, followed by choosing

all the necessary characters for recognition from each language.

3. ***Word Dictionary Preparation and Creation.*** Any language integrated into FR12 comes with its own vocabulary. If we create a new language, i.e., the user's language, a specific vocabulary for that language needs to be created. For example, for documents in Romanian printed with Romanian Cyrillic letters, we need Romanian words in Cyrillic script. One solution would be to transliterate words from the modern (Latin) script to the Cyrillic script and add them to the dictionary in FR 12.

4. ***Template Creation and Training.*** Each individual character is machine-learned in a supervised manner (involving specialists), so that the segmented characters from the image are matched with their corresponding Unicode digital characters. Following the training of the OCR engine in FR, typically on a volume of 2-10 pages of the researched document, a characteristic template (model) is created for that specific document, or generally, characteristic of the typography in which the document was printed. The number of pages required to achieve a quality result depends on the specific nature of the document. In the learning process, FR uses not only the established results from processing the respective document, but also a series of previously trained (pre-trained) neural network models, as well as incorporated mechanisms based on statistical analysis, thereby improving the recognition process.

Contextual information plays a significant role and is used in the OCR engine in a manner similar to human text reading, with words often being predicted after only a few characters, taking into account the context, i.e., the meaning of the entire sentence. However, the gained experience is not shared between different documents, which is a temporary disadvantage. In future versions of FR, advanced neural network architectures such as recurrent neural network architectures may be introduced, which could potentially link the recognition experience (memory) between pages or even between documents.

Thus, out of the steps listed above, only the first one - image preprocessing - will be practically used without modifications using existing software. For the remaining steps, developments are needed to extend the preset capabilities of the existing software.

In the following section, we will describe the properties and functionality of the image preprocessing modules, highlighting their specific application when dealing with old texts.

4.3 User Language Creation and Dictionary Addition

Adding the alphabet and creating the user language are the extended features of the ABBYY FineReader software.

However, these features are not full functioning for non standard alphabets. When processing the Romanian Cyrillic alphabet of the 17th century, the main difficulty is that the characters Ѧ ѧ and Ѵ ѵ do not exist in the alphabet addition system in FR 12 and cannot be displayed by its system fonts. Therefore, it was necessary to adapt them from specialized software like BabelMap2. The rest of Romanian Cyrillic characters can be selected through the alphabet window in FR 12, as shown in Fig. 4.2, p. 55.

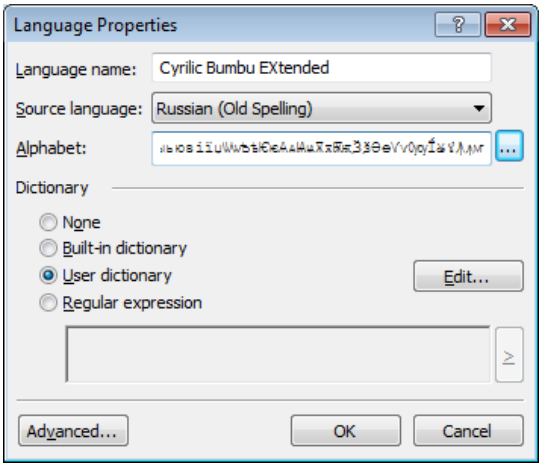


Figure 4.2: User Language Creation Window in FR 12

The word dictionary can be expanded using three methods.

The **first method** involves transliterating existing vocabularies from the modern (Latin) alphabet to the Cyrillic one, and adding them to the dictionary in FR 12. This method partially solves the problem because many words from the 17th and 18th centuries are no longer present in the modern Romanian vocabulary, and vice versa, many words from the modern vocabulary would be unnecessary in a system that recognizes documents printed in the 17th century.

The **second method** involves creating the dictionary from the already recognized document. It means, that all the words in the text obtained after OCR are manually corrected and added to the „user dictionary" as shown in Fig. 4.3, p. 56.

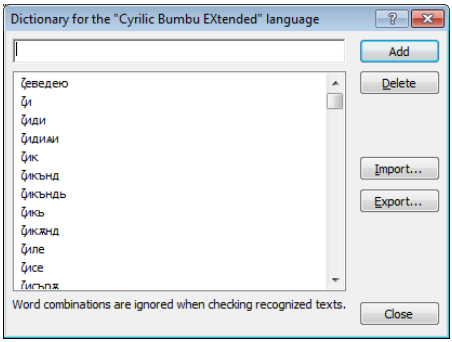


Figure 4.3: Word Dictionary Addition Window in FR 12

The **third method** of adding words to the user dictionary is also based on the fact that some portions of the document have been recognized. From the window with the recognized text (Fig. 4.4, p. 57), during the spell-checking process, there is the possibility to include the word underlined in red using "Add to Dictionary" right-click option.

4.4 Training Process with FR12, Template Creation

A template is a set of pairs consisting of the printed character and the corresponding digital character used by the computer. Templates can be created during training.

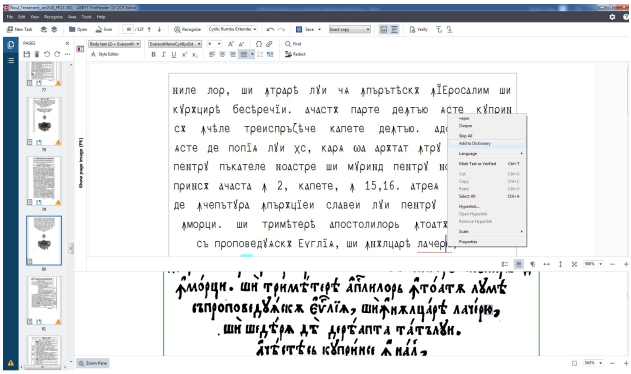


Figure 4.4: FR 15 Window with Recognized Text

The training process of FR12 with Romanian documents printed in the 17th century can be carried out as follows:

• several pages of the document are recognized through the supervised training procedure presented in Fig. 4.5, p. 57,

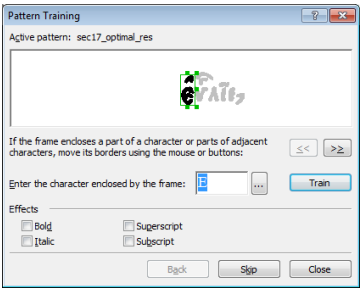


Figure 4.5: FR Template Learning Window

• template is created as pairs of the „image character" (character cut from the preprocessed document image) and the „digital character" (UNICODE character) as shown in Fig. 4.6, p. 58.

• the resulted template is used as an additional source of information to aid in recognizing the remaining text.

Sometimes, two or more characters can be joined together, making it

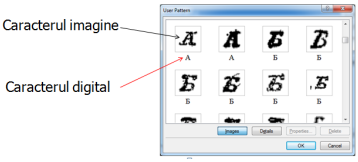


Figure 4.6: Window with pairs: 'image character', 'digital character'

practically impossible to recognize each character separately. In this case, they will be recognized together in a single bounding box, as a composite character, known as a ligature. In 17th-century Romanian documents, we encounter a more distinctive type of ligatures, namely „vertical ligatures" - a set of characters that do not explicitly join but are stacked one above the other (see Fig. 4.7, p. 58). Examples of such combinations are very common in the studied documents.

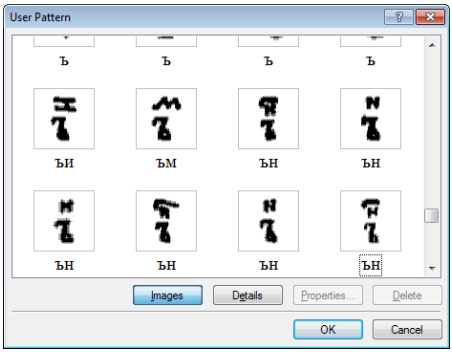


Figure 4.7: Sets of Ligatures

4.5 OCR Models Applied to Texts Printed in the 17th Century

The case study of optical character recognition of 17th-century books, described in this chapter, was conducted on the book „New Testament" printed in 1648, in Balgrad (Alba Iulia), using black and red ink, with 34 lines per page.

The first cover is adorned with a border featuring vegetal motifs, created through embossing. In the center, the Crucifixion scene is depicted.

The New Testament contains 682 pages, and the first approximately 270 pages comprise the four Gospels:

- The Gospel according to Matthew;
- The Gospel according to Mark;
- The Gospel according to Luke;
- The Gospel according to John;

Optical Character Recognition (OCR) was applied to the Gospels according to Matthew, Mark, Luke, and John, which together constitute 267 pages. After preprocessing with Scan Tailor, the default preprocessing in FR 12 was run. A common aspect of books printed in the 17th century is the writing of certain letters above other letters, as exemplified in Fig. 4.8, p. 59. This may be due to the fact that in the printing process, some letters were omitted, and then they were written above the preceding letter. The most frequently omitted consonants are:с, д, ш, н, м, л, к, р, х, ц, ч, and vowels: а, и, ь.

Figure 4.8: Representation of vertical ligatures in the document

Furthermore, abbreviations using tildes or other diacritical marks are used(Fig. 4.9,p. 59). Abbreviations were used especially for proper names, such as Jesus ЇС), God Ѧмнєzхѵ). Taking this aspect into consideration, we will significantly increase the resolution (over 1200 DPI) so that we have the entire frame of a character in the training process..

Figure 4.9: Representation of the abbreviation ЇС (Isus) in the document

The numbers are also represented by letters with tildes above them (Fig. 4.10, p. 60).

Figure 4.10: Representation of the number ”24”

In the process of training the templates, textual numbers have been replaced with Arabic numerals (Fig. 4.11, p. 60).

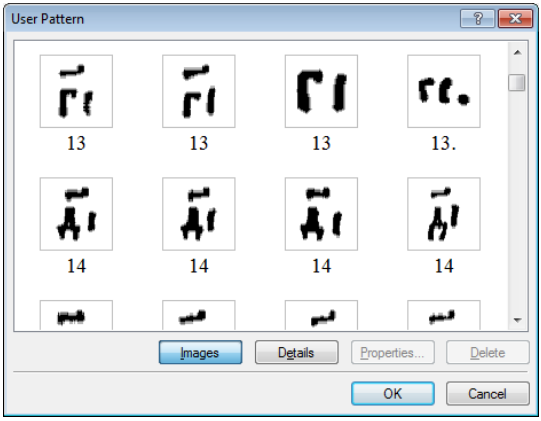


Figure 4.11: Template for recognizing numbers in the New Testament printed in 1648 in Balgrad

Another characteristic of books and documents from the 17th century is that many words are printed together. Most of these compound words are combinations of prepositions/articles joined with other parts of speech. In Fig. 4.12, p. 61, an example of a compound word is shown (**алдонлє** equivalent to the modern expression **al doilea**). This complicates the process of expanding the dictionary used to support optical recognition. In this regard, it would be advisable for all these words to be separated. To automate this process, a small program was created that separates these words. This program is based on a manually created vocabulary from already recognized documents.

At present, the dictionary of words used for optical character recognition for texts from the 17th century contains more than 7500 words.

Figure 4.12: Examples of words written together (encircled)

Taking into account that in the 17th century, 47 characters of the Romanian Cyrillic alphabet were used for printing, in the training stage, we need a keyboard that will contain all these characters. As a result, a virtual keyboard (see Fig. 4.13,p. 61) was developed for Windows 7, 8, 10 operating systems using the tool Microsoft Keyboard Layout Creator3, integrating all these characters.

Figure 4.13: Virtual keyboard with the Romanian Cyrillic alphabet

In this subsection, some characteristic properties of recognizing documents from the 17th century were described. In the next subsection, we will describe the process and results of OCR evaluation for Romanian Cyrillic documents from the 17th century.

4.6 OCR Evaluation of 17th Century Documents

For the OCR evaluation of 17th century documents printed with Romanian Cyrillic characters, we will use ABBYY FineReader PDF 15 OCR Editor (referred to as FR 15).

For this purpose, a dataset with pages from the New Testament (1648) was prepared. For the training set, 10 pages were selected. For the test set, 5 pages were selected, which include approximately 7090 characters and about 1280 words. On average, a single page from the test set contains 1400 characters and 260 words. These pages have already been manually recognized, verified,

and corrected in a previous training process, which was not subjected to a strict accuracy measurement. Additionally, for some experiments, we used a dictionary consisting of 4582 words extracted earlier from recognized pages from the 17th and 18th centuries. The words in the dictionary were checked and corrected before being added to FR 15.

In this evaluation, the criteria taken into account were OCR accuracy both at the character level and at the word level. The evaluation results for a specific number of test/training pages will be summarized to obtain an overall OCR accuracy. The overall accuracy (OA) for a specific experiment will be calculated using the formula4.1 described in reference [40]:

$$AG = \frac{\sum_1^n c_i}{\sum_1^n t_i} \tag{4.1}$$

where n is the number of test pages for each experiment, is the number of correctly recognized characters or words from an arbitrary page, and is the total number of characters/words in the test set used in the experiments.

The OCR evaluation results are presented in Table 4.1 p. 64. Each row in the table represents an experiment conducted on a specific training and testing dataset consisting of a defined number of pages. The table is composed of the following columns:

• Experiment - which includes the number of pages used for training and the number of pages used for testing;

• OCR Accuracy at the character level with the use of the word dictionary;

• OCR Accuracy at the word level with the use of the word dictionary;

• OCR Accuracy at the character level without the use of a word dictionary;

• OCR Accuracy at the word level without the use of a word dictionary.

We will mention that accuracy evaluation is performed through two methods. The first method is manual and involves a person counting the incorrect characters/words on a single page from the test set. The second method automatically compares the differences between the document verified beforehand and the document recognized by the newly trained model.

Next, we will iteratively train a model with 1, 2, 5, 7 pages. Additionally, at each stage, we will test the trained model together with the word dictionary,

but we will also conduct experiments without involving the dictionary in the recognition process.

First Experiment: We trained the FR 15 model with one page from the training set. From this page, 662 glyphs were extracted in the training process using the GUI interface provided by FR 15. The most frequent characters are 'и' and 'a', which occurred 51 and 48 times respectively, while the least frequent glyph is the character 'ѳ', which occurred only once in the training set. In this iteration, we evaluated the OCR model, both with and without the word dictionary. With the dictionary, we counted 132 incorrect characters out of a total of 1460 characters, and out of 256 words in total, 69 had at least one incorrect character. Without the word dictionary, we observed the presence of 171 incorrect characters and 108 words with at least one incorrect character.

Second Experiment: We trained the FR 15 model with 2 pages from the training set. From these pages, 1275 glyphs were extracted. The glyph 'a' appears 95 times in this training set, and on average, there are over 30 glyphs per character. When recognizing a page from the test set with the word dictionary, we counted 99 incorrect characters out of 1460 characters, and 56 words had at least one incorrect character. Without the word dictionary, 134 incorrect characters were identified, and 76 words out of a total of 256 were incorrect.

Third Experiment: We continued by training the OCR model with 5 pages from the training set, containing over 2500 glyphs. When recognizing with the word dictionary, out of 1460 tested characters, 73 were recognized incorrectly, and out of 256 evaluated words, 50 were found to be incorrect. Without the word dictionary, 86 incorrectly identified characters were observed, as well as 69 words with errors. The majority of errors in this experiment are caused by ligatures, with 44 out of 56 ligatures being incorrect.

Fourth Experiment: The OCR model was trained with 7 pages from the training set. The number of glyphs in the set is over 3600. With the word dictionary, 59 characters out of 1460 were recognized incorrectly, and 52 out of 256 words had at least one incorrect character. Compared to Experiment 3, the character-level accuracy gained in this experiment continued to increase by approximately 1%, but the word-level accuracy remained at the same level.

In fact, we have 2 more incorrect words than in the previous experiment when using the dictionary.

Experiments			With dict.		Without dict.	
Nr.	Antrenare	Testing	A_{ch}	A_{cuv}	A_{ch}	A_{cuv}
1	1 page (662 glyphs);	1 page;	0.91	0.73	0.88	0.57
2	2 pages (1275 glyphs);	1 page;	0.93	0.78	0.90	0.70
3	5 pages (2540 glyphs);	1 page;	0.95	0.80	0.94	0.73
4	7 pages (3668 glyphs);	1 page;	0.96	0.796	0.95	0.75

Table 4.1: OCR Accuracy in Recognizing 17th Century Documents

The training and evaluation mode of the OCR model using FR 15 for the recognition of Romanian Cyrillic characters from the 17th century, presented above, allows us to conclude that the respective software system contains the major components necessary for training, making it possible to apply it also for the case of old Romanian printings (in our case – from the 17th century), which were not foreseen in the initial set. An approach based on such tools equipped with a graphical user interface allows also non-advanced users to train OCR engines and to participate in digitization projects.

The training approach was evaluated using several training and testing pages in an iterative process, from one training page up to seven pages. As the number of training data increased, significant improvements in the accuracy of the model were observed. Examples of recognition, transliteration, and alignment of a 17th-century document are presented in Fig. 4.14, p. 65, Fig. 4.15, p. 65, Fig. 4.16, p. 65, Fig. 4.17, p. 66.

4.7 Classification of 17th Century Fonts

The printing presses of the 17th century used several fonts or, more precisely, several sets of distinct characters in the printing of documents. However, of these, for now, two completely different fonts stand out, both in the style of writing/printing and in the use of characters [31],[32].

The problem of identifying the font in a document printed in the 17th century can be formulated as follows: Given a document X printed in the

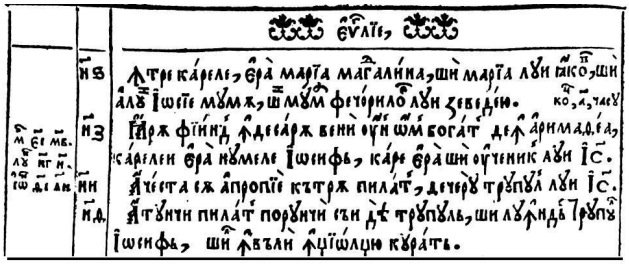


Figure 4.14: Fragment from the New Testament printed in the year 1648 in Balgrad

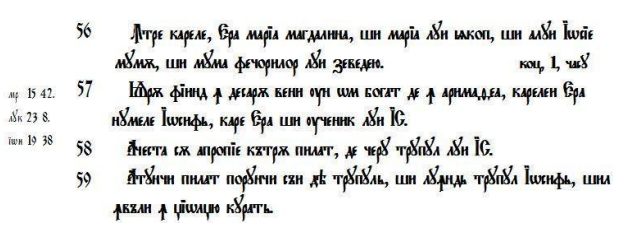


Figure 4.15: The document from Fig.4.14 following the OCR stage

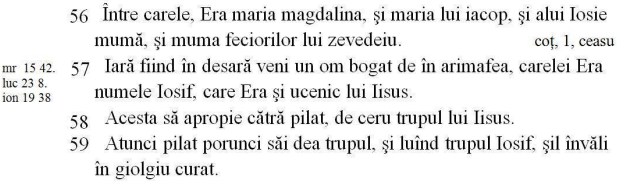


Figure 4.16: The text from Fig.4.15 after the transliteration stage

17th century with Romanian Cyrillic characters and a set N of OCR models trained on documents from this period. The task is to choose the most suitable model M from set N for the recognition of document X [32].

A solution would be to recognize a sample (a page/a page fragment) from

56. Intre care erau Maria Magdalena si Maria, mama lui Iacob si a lui Iosif, si mama fiilor lui Zevedeu.
57. Iar in amurgul zilei a venit un om bogat din Arimateea, cu numele Iosif, care era si el ucenic al lui Iisus.
58. Acesta, ducandu-se la Pilat, a cerut trupul lui Iisus. Atunci Pilat a poruncit sa i se dea.
59. Si Iosif, luand trupul, l-a infasurat in giulgiu curat.

Figure 4.17: Fragment from the Gospel according to Matthew, the current version aligned with the text from Fig.4.16

document X with all the templates trained in FR12 on documents from the 17th century and based on the obtained results, to choose the template that provides the highest accuracy. This solution is easy to implement, but the time complexity is too high, as we need to load each model M in turn. The loading time of a model plus the recognition of the sample can exceed 2 minutes depending on the page size. For 5 different OCR models, we would have to wait approximately 10 minutes to find the most suitable model M.

Given that the number of the first printing houses was not too large, we could approach another direct solution, where all OCR models are classified according to Printing Houses, and in which the user can choose the printing house. This solution has been implemented and is described in the following section.

4.7.1 OCR Model Selection Program Based on Printing House

In this program [31], the user can choose the century, and from the following options, they might select one of the regions where the Romanian Cyrillic script was used, namely: Iași, Bucharest, Târgoviște, Alba Iulia (Bălgrad), Chernivtsi (Uniev), Sebeș (Sas Sebeș), Snagov, Buzău.

We will proceed with the selection of the printing house from one of these regions.

Thus, for **Iași**, the following printing houses are available:

1. Tipariul cel Domnesc/ The Royal Printing House;

2. Casa Sfintei Mitropolii/ The House of the Holy Metropolitan Church;
3. Tiparnița Tărâi/ The Country's Printing Press;

In **Bucharest**, the following printing houses were present:

1. Scaunul Mitropolii Bucureștilor/ The Seat of the Metropolitan of Bucharest;
2. Tipografia Domnească/ The Royal Typography; In **Balgrad** the following printing presses were operational:
1. Tipografia Domnească/ The Royal Typography;
2. The Metropolitanate of Alba Iulia. The **other regions** had only one printing press each; we will present their names in the original form:

• Târgoviște – Sfânta Mitropolie a Târgoviştii/ Târgoviște – The Holy Metropolitanate of Târgoviște;

• Uniev – Sfînta Mănăstire Uniev/ Uniev – The Holy Monastery of Uniev;

•Sas Sebeș – Tipograf[ia] Noao/ Sas Sebeș – The New Printing House;

• Snagov - Tipografiia Domnească în Sfânta Mănăstire în Snagov/ Snagov – The Royal Printing House at the Holy Monastery in Snagov;

• Buzău - Tipografiia Domnească, la Episcupiia dela Buzău/ Buzău – The Royal Printing House at the Episcopal See of Buzău;

Documents and books that were printed in these printers will be able to be recognized using the most suitable template. The main window of FR 12 will open with the set template, and further the optical character recognition process will take place. Next, Fig. 4.18, p. 68 shows the graphical interface of the application for selecting the most suitable recognition template. The "Model Selector" application is written entirely in Java and works with any installed FR version.

The third solution would be to train a neural network to classify a sample from document X based on samples from multiple Romanian documents printed in the years 1640-1700 at various printing houses. We will use a neural network that will employ a dataset consisting of pairs in the form of <character from the image, class>. We will describe the implementation of this solution further.

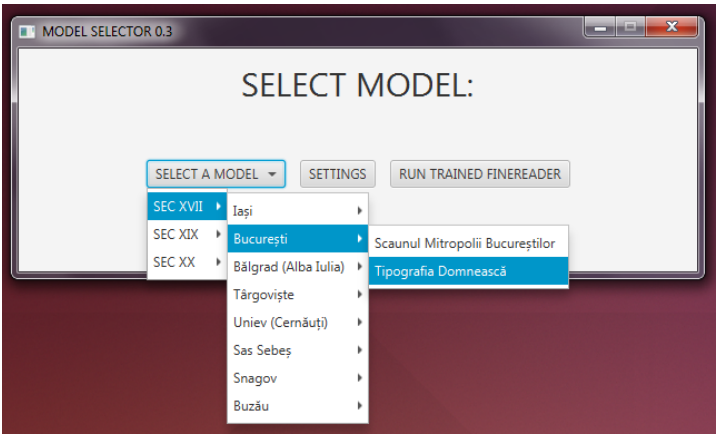


Figure 4.18: The main window of the Model Selector application

In Fig. 4.19, p. 69 , we can observe two pages from two books from the 17th century that were printed with two different fonts - with two different sets of characters. It is noticeable that the two texts have different styles and, besides this, characters with very distinct shapes are used. If we choose a template that was trained on the first text and apply it to the second text displayed in figure 5.17, then we will obtain a result with a high error rate. In this case, the most appropriate solution would be to create a new template trained on the second text. In what follows, we will describe how to address this problem by creating a neural network, trained to perform classification of two distinct fonts used in the 17th century for printing documents in Romanian.

4.7.2 Classifying fonts using neural networks

In the following, we will describe the solution to the problem of classifying characters from the 17th century. It's worth noting that there are several useful free tools available that can contribute to the development of this chapter. And, of course, one of the main aspects specific to any application based on neural networks is the creation of a dataset for training machine learning algorithms.

Figure 4.19: Texts printed in the 17th century in Romanian Cyrillic alphabet at two different printing houses, each with distinct fonts [32]

Creating the dataset. The dataset will be created from 10 scanned books selected from the Digital Library of Romania, which are as follows:

1. „Noulu Testamentu sau Înpacarea" tipărit în 1648 în Cetatea Bălgradului. /„The New Testament or The Reconciliation" printed in 1648 in the City of Bălgrad.

2. „Liturghie şi rugăciuni" tipărită în 1683 în Casa Sfintei Mitropolii de la Iaşi de mitropolitul Dosoftei. / „Liturgy and Prayers" printed in 1683 in the House of the Holy Metropolitan Church of Iași by Metropolitan Dosoftei.

3. „Parimiile preste an" tiparită în 1683 în Tiparniţa Ţărâi de la Iaşi. / „The Prayers Throughout the Year" printed in 1683 at the Country Printing House in Iași.

4. „Psaltirea a proorocului şi înpăratului D[a]v[i]dŭ Cu m[o]l[i]tve la toate Cathizmele Şi cu pashalii de 50 de ani. După orânduiala grecească. Şi la săvârşitŭ exapsalmu" tipărită în 1694 la Tipografia Domnească în Sfânta Mitropolie din București. /„The Psalter of the Prophet and King David with prayers for all Catechisms and with paschal prayers for 50 years. According to the Greek rite. And in the end, exapsalm" printed in 1694 at the Royal Printing House in the Holy Metropolitan Church of Bucharest.

5. „Carte sau lumină cu dreapte dovediri din dogmele Besearicii Răsăritului, asupra dejghinării Papistaşilorŭ" tipărită în 1699 în Tipografia Domnească în Sfânta Mănăstire în Snagov. „Book or Light with Right Proofs from the

Dogmas of the Eastern Church Concerning the Thawing of the Papists" printed in 1699 at the Royal Printing House in the Holy Monastery of Snagov.

6. „Pravoslavnica mărturisire a săborniceştii, şi apostoleştii Besearecii Răsăritului Dupre Grecească" tipărită în 1691 În tipografia Domnească, la Episcupiia de la Buzău de Petru Movilă, mitropolit al Kievului. / „The Orthodox Confession of the Catholic and Apostolic Church of the East According to the Greek Rite" printed in 1691 at the Royal Printing House, Episcopal See of Buzău, by Petru Movilă, Metropolitan of Kiev.

7. „Septembrie-Decembrie Vol. 1" tipărită în 1682 în Tipografia Sfintei Mitropolii de la Iași. / „September-December Vol. 1" printed in 1682 at the Printing House of the Holy Metropolitan Church of Iași.

8. „Mineiulŭ. Luna lui Dechemvrie" tipărită în 1698 în Sfânta Episcopie de la Buzău. / „The Minei. The Month of December" printed in 1698 at the Holy Episcopate of Buzău.

9. „Apostolul cu Dumnezău Svântul Carea întru acesta chip tocmită depre orânduiala grecescului Apostol" tipărită în 1683 în Scaunul Mitropolii Bucureştilor. / „The Apostle with God the Holy Carea thus arranged by the priests according to the greek Apostolic order„ printed in 1683 at the Seat of the Metropolitan of Bucharest.

10. „Sfânta şi Dumnezeiasca Evanghelie" tipărită în 1697 în Sfânta Mănăstire în Sneagovŭ din Snagov. / „The Holy and Divine Gospel" printed in 1697 at the Holy Monastery in Snagov from Snagov. In the ten selected books, two distinct sets of characters used for printing were observed, which we will refer to as fonts A and B. Thus, the books used for forming the dataset were divided into two groups: books numbered 1, 8, 9, and 10 were grouped in the set with font A, and books numbered 2, 3, 4, 5, 6, and 7 were grouped in set B (the second font). In the formation of the dataset, a total of 22 pages were processed, including 13 pages extracted for set A and 9 pages for set B. In Figure 5.18, two collages composed of selected pages are presented.

In the ten selected books, two distinct sets of characters used for printing were observed, which we will refer to as fonts A and B. Thus, the books used for forming the dataset were divided into two groups: books numbered 1, 8, 9, and 10 were grouped in the set with font A, and books numbered 2, 3, 4, 5, 6, and 7 were grouped in set B (the second font). In the formation of the

Figure 4.20: Collage from books 1, 8, 9, 10 with character set A

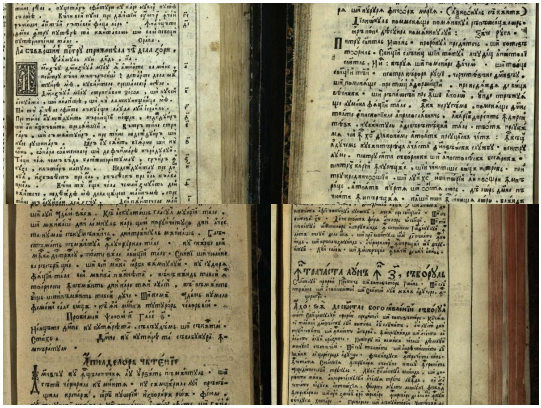


Figure 4.21: Collage from books 2, 3, 4, 7 with character set B

dataset, a total of 22 pages were processed, including 13 pages extracted for set A and 9 pages for set B. In Fig. 4.20, p. 71 and Fig. 4.21, p. 71 two collages composed of selected pages are presented.

To address the problem of font classification, it is necessary to perform the following actions:

1. Segmentation and cropping of text blocks from the selected pages.
2. Detection of individual characters within the text blocks.
3. Creating the training and testing dataset from the characters detected in step number 2 through clustering/grouping of the extracted characters.
4. Training a multi-layer neural network (MLP) to classify the characters and evaluating it.
5. Utilizing the trained algorithm (model) to classify characters in a new image.

In the following, we will describe in detail the process of executing the tasks listed above.

Task 1. Segmentation and cropping of text blocks

From the prepared pages for the dataset, we will segment and crop text fragments using Detectron24, a platform for object detection, segmentation, and other visual recognition tasks developed by Facebook, as well as the document segmentation model based on the PrimaLayout5 dataset with the *mask_rcnn_R_50_FPN_3X* configuration implemented in the LayouParser6 package.

In the model with the PrimaLayout dataset, text portions are labeled with the „TextRegion" label, (region/text block) which has the ID 1. Based on this ID, we will select the text blocks.. Considering the complexity of the layout in 17th-century documents, such as titles, verse numbers in a new line, numbers written in Cyrillic letters, chapter headings, etc., we will segment and cut more than one text block from a single page. For this reason, it is possible that two text blocks may contain the same characters, and as a result, our dataset may contain similar training and testing examples. This is not a problem at the initial stage, and the optimization of the dataset can be done later in step 3 when creating the training and testing dataset.

When applying the segmentation tool to the selected pages, we will obtain portions with text blocks that will help us crop the necessary fragments. An example of segmentation is shown in Fig. 4.22, p. 73. From this image, we can see that two rectangles have been outlined, intersecting and delineating two text blocks, even though the page/image is actually composed of a single

block. However, as mentioned earlier, at the initial stage, the repetition of the same characters from different text blocks is not a hindrance.

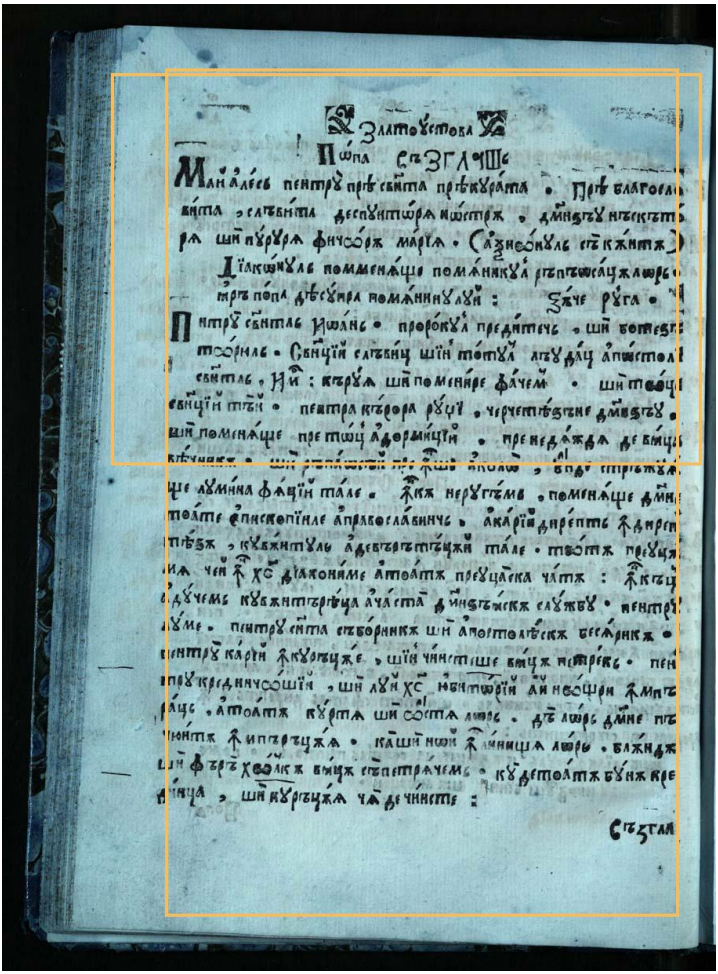


Figure 4.22: A page from book number 2 that has been segmented with PrimaLayout

After segmentation, the text blocks need to be cropped and placed into

a list of text block fragments. On average, four text block image fragments were obtained for each of the 22 pages. In the next step, from these fragments, we will identify and extract characters through cropping. Characters include letters, punctuation marks, accents, some lines outlining tables, pixels of stains or page damage.

Task 2. The detection of individual characters from the text blocks

So, the next task is to find a way to extract characters from the text block fragments. In the next step (step 3), we can then create a dataset from the extracted letters and ultimately train a classifier on this dataset.

Inputs into our system are pages from documents printed in the 17th century. These inputs can be images from various sources (smartphone or digital camera, scanner, etc.), with different resolutions. Our goal is to process each image, regardless of its source, in a way that the character detection algorithm can find as many letters as possible. It's worth noting that digital cameras store images in three separate channels: red, green, and blue (RGB). In our case, these three channels contain redundant information because letters can be identified in each of these three channels separately. Therefore, we will first convert all images to grayscale, so instead of three channels, we work with just one, which should increase the learning speed. As a result, the processed image will consist of only black and white pixels. We can then further optimize the image for letter detection. In addition to existing software, a Python function has been developed to convert images from RGB to grayscale using the OpenCV library.

To detect characters, we will use the findContours() method from the OpenCV library. We can then map the bounding boxes of the contours found by this function back onto the original RGB image to see what was actually detected. The result of this processing step is exemplified in Fig. 4.23, p. 75.

The identified characters have been cropped and saved in two folders, one for each font. From the 10 pages in set A, we obtained 17,155 characters, while from the 9 pages in set B, 8,799 characters were saved. The difference in the number of extracted characters for sets A and B can be explained by the sizes of the characters used in the two distinct fonts, the number of characters per page, and the number of cropped text block fragments. Among the extracted characters, some may be noise elements from the image, such

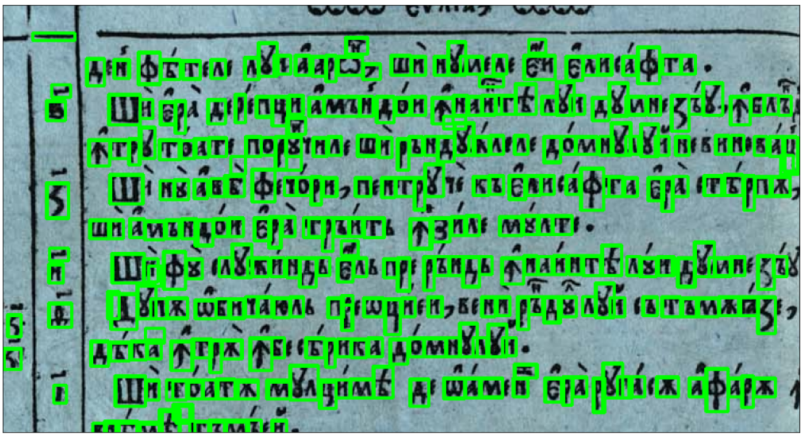


Figure 4.23: Identification of character contours in a text block fragment

as stains, accents without the base character (letter), tildes, etc.

In the next step, we will remove unnecessary characters and keep only the letters using the K-Means7 clustering model and Principal Component Analysis8 (PCA).

Task 3. Clustering characters for the creation of the training and testing dataset

The next task to be accomplished is to eliminate images that do not contain letters and cluster all the remaining images (meaning that all the letters from collection "A" go into one folder, and all the letters "B" go into another folder). To obtain a high-quality dataset, we will combine manual processing with automated data grouping using PCA and K-Means.

PCA and K-Means methods address different tasks. PCA is used for dimensionality reduction or feature selection when the feature space contains too many irrelevant or redundant features. Its goal is to find the intrinsic dimensionality of the data. On the other hand, K-Means is a clustering algorithm that returns the natural grouping of character vectors based on their similarity.

Before beginning clustering, we need to ensure that all images have the same dimensions. As seen in Figure 5.20, the characters have been saved with

different dimensions. Each image has the size of its bounding box, which varies widely. So, in the first step, we will resize all images to 50x50 pixels and concatenate them into a matrix using the NumPy9 library. Then, we will need to reduce the dimensionality of the characters. Each image contains 50x50 (2500) pixels or features. This quantity is too large for a clustering algorithm, so we will use Principal Component Analysis (PCA) to reduce the dimensionality from 2500 to 25.

After these steps, the data is prepared to be clustered so that the unlabeled dataset is turned into a labeled state. We will apply K-Means clustering, which is a simple and fast method. The only thing we need to provide is the number of clusters we expect. If K-Means were to give us a perfect result, we would get exactly the number of letters in the Romanian Cyrillic alphabet - 47, but it's less likely to achieve such an ideal result, considering variations in the differences between uppercase and lowercase letters. Each of the 47 letters can also appear in both uppercase and lowercase forms, which means we could expect a total of 47x2 (94) clusters. There will also be punctuation marks and noise in the data, which may have been detected. Therefore, it is reasonable to set the output parameter to 100 clusters. This is more than we need, but it will be easier to merge clusters later than to separate them.

The clustering process for the approximately 25,000 characters took less than a minute, and the result of clustering will be moved into separate folders based on cluster labels (Fig. 4.24, p. 76).

Figure 4.24: Folders with characters grouped into cluster

After automatic clustering, manual processing is performed, which in-

cludes the following actions: moving misclassified images to the correct folders, merging folders with identical letters, identifying noises, etc.

Some of the resulted clusters are composed of parts of characters (angles, curves, lines) (Fig. 4.25, p. 77).

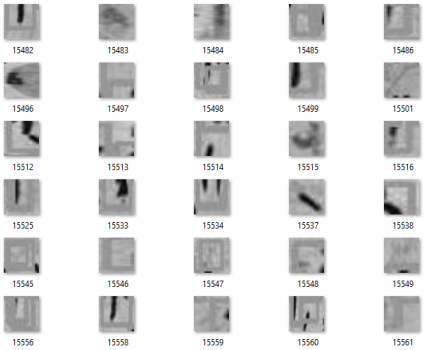


Figure 4.25: Cluster with parts of characters

Another part of the resulted clusters is formed by letters of the Romanian Cyrillic alphabet. For example, in Fig. 4.26, p. 77 the cluster is formed of the font B letter ⷒ (the letter t in the Latin), in Fig. 4.27, p. 78 - by the glyphs of the font A letter ⷜ(the combination of letters 'ea' in the Latin).

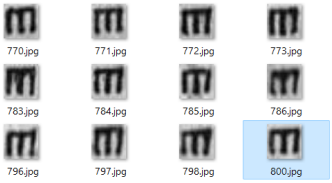


Figure 4.26: Cluster formed by the glyphs of the letter ⷒ

After the manual cleaning and adjustment of clusters, we will place the images from them into two folders, one for the set with font A and the other for characters with the font represented by set B. The result of this process is two folders named *fontA* and *fontB*. In the *fontA* folder, there are over 21,000 characters, and the *fontB* folder contains over 8,700. The imbalance in the

Figure 4.27: Cluster formed by the glyphs of the letter ꝉ

number of examples for each of the two fonts can be managed in the dataset splitting stage if we encounter low accuracy in classification.

The final step to complete the formation of a well-organized dataset will be to convert the dataset into IDX10 format, a format known for its use in presenting the well-known MNIST11 handwritten digits database. We will do this using the *idx_converter()* function from the *idx_tools* package, which takes a file structure directly from the operating system and saves it in IDX format. Since we want to train a neural network, we should split the images into a training and testing dataset. For this, we will move 30% of the images from each set A and B to a testing folder. The output will consist of 4 files: 2 files with images for training and testing, and another 2 files with the corresponding labels for the image files. The class labels for the characters will be values 0 and 1, where 1 represents the class of characters from fontA, and 0 represents the class of characters from fontB. After these operations, we will have over 21,200 training examples and over 9,000 testing examples.

Next, we will train a neural network (NN) using the dataset prepared up to this stage.

Task 4. Training a NN for character classification and evaluating it

We will train a multilayer neural network (MNN) on the dataset prepared in the previous stage to classify characters into the two different fonts. This approach will follow the example from the Tensorflow12 website, but the architecture of the MNN model will be tailored for binary classification.

First, we need to load the data we saved earlier in the IDX data format. We have already split it into a training and a test dataset, and each of these

two datasets comes with a label file that we will load. Fig. 4.28, p. 79 shows some random examples to verify that the dataset has been saved correctly.

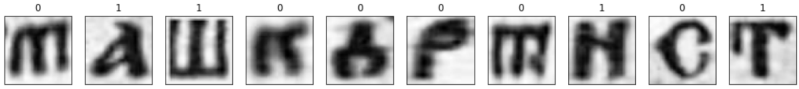


Figure 4.28: Random examples from the training set [32]

Before we begin constructing the MNN architecture, we need to normalize all the images so that pixel values are real values between 0 and 1 instead of 0 to 255. To do this, we will scale the pixel values in the images to 255.

The multilayer neural network described here (Fig. 4.29, p. 79) is structured into three main layers: an input layer, a hidden layer, and an output layer. The input layer consists of neurons representing each distinct feature

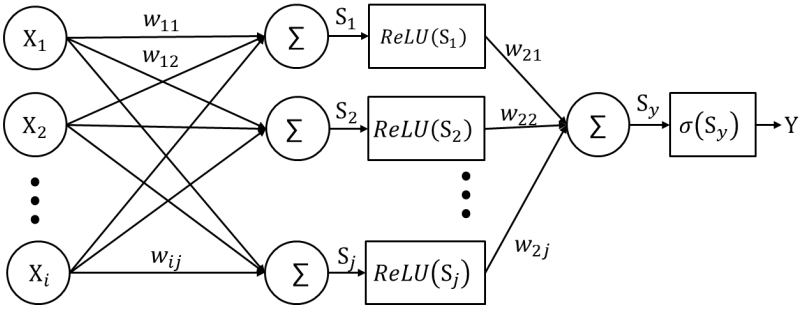


Figure 4.29: Neural network architecture for font classification

X_i of the character in the image. In this case, there are 2,500 such features being represented. The hidden layer of this network consists of 128 neurons and uses the (Rectified Linear Unit) ReLU13 activation function. ReLU is a popular activation function in neural networks because it does not saturate to a fixed upper bound during backpropagation, making network training more efficient. The output layer of the network contains a single neuron,

using a sigmoid activation function σ defined as

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{4.2}$$

where x is the weighted sum of the outputs S_y from the hidden layer. Sigmoid functions have an „S" shape and are often used in neural networks for binary classification because they produce an output between 0 and 1, which can be interpreted as a probability.

The construction of the neural network begins with transforming the input data from a matrix of size x by y (in our case, 50*50) into a vector of length. This is done using the *keras.layers.Flatten14* layer. Then we add 128 neurons in the hidden layer, which is fully connected to the last layer. A single neuron in the output layer is sufficient because we have binary classification. The neural network was trained over 300 epochs and took approximately 55 minutes without using a graphics processing unit (GPU15).

The training results show an accuracy of 96.7% (Fig. 4.30, p. 80).

```
Epoch 295/300
21212/21212 [==============================] - 7s 348us/sample - loss: 0.0458 - accuracy: 0.9864
Epoch 296/300
21212/21212 [==============================] - 8s 357us/sample - loss: 0.0486 - accuracy: 0.9847
Epoch 297/300
21212/21212 [==============================] - 8s 359us/sample - loss: 0.0455 - accuracy: 0.9862
Epoch 298/300
21212/21212 [==============================] - 8s 364us/sample - loss: 0.0469 - accuracy: 0.9853
Epoch 299/300
21212/21212 [==============================] - 7s 353us/sample - loss: 0.0461 - accuracy: 0.9851
Epoch 300/300
21212/21212 [==============================] - 8s 366us/sample - loss: 0.0456 - accuracy: 0.9860
```

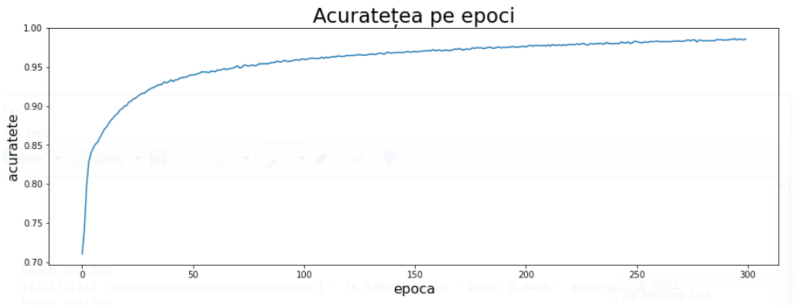


Figure 4.30: The accuracy history for each epoch individually

The confusion matrix, which shows the model's predictions compared to the actual classes, is presented in Fig. 4.31, p. 81.

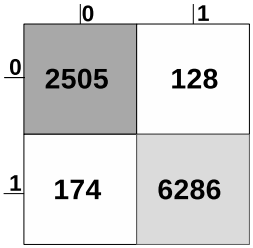


Figure 4.31: The illustration of the confusion matrix on the test dataset

If we calculate the classification error using the false positives and false negatives from the confusion matrix, we get a $(128 + 174) * 100\%/9093 = 3.3\%$ error rate.

We will save the MNN model structure in a json file and the network weights in an HDF516 file. When using the model, we will input a set of characters from pages with text blocks in Romanian Cyrillic alphabet printed in the 17th century. We will classify the document based on the class that obtained the highest number of predictions. After classifying the document, FR 12 will be used with the corresponding OCR model for the document's font.

4.8 Post-OCR text improvement

The improvement of the recognized text following OCR has been extensively discussed in [121]. In our experimentation with post-OCR spell checking for old Romanian texts, a key challenge arose: the absence of dictionaries tailored to old Romanian from various centuries. Consequently, our strategy involved leveraging existing software and substituting the dictionary.

The choice between commercial software (e.g., MS Word) and free Open

Source alternatives (such as LibreOffice Writer) was considered. All spell checkers support user dictionaries, making the process seemingly straightforward: designate the text's language with installed proofing tools and compile the user dictionary with essential words.

In MS Word, this approach yielded partial success (Fig. 4.32). Certain old Romanian Cyrillic letters, such as ѧ, ѵ, ѡ, and the ligature **ia**, were not recognized as letters in the spell checker (Fig. 4.33). Transliteration of these *no-letters* may resolve the issue.

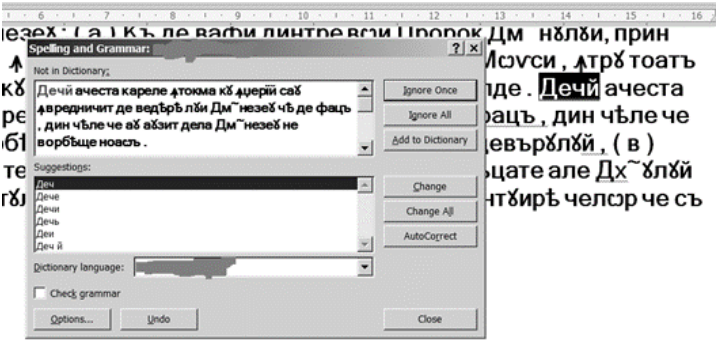


Figure 4.32: Successful work of MS Word over an old Romanian text

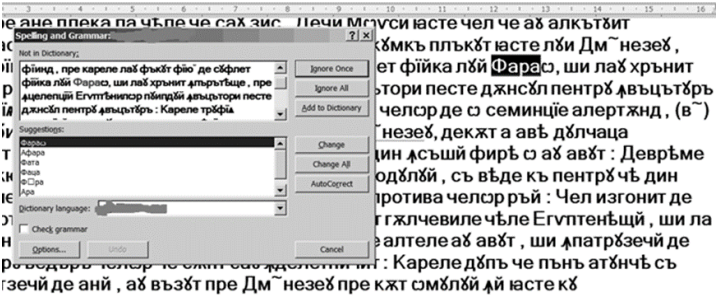


Figure 4.33: Invalid work of MS Word over an old Romanian text

In contrast, using LibreOffice Writer under Linux SuSe proved successful in our experiment, particularly in testing the aforementioned *no-letters*. New

words could be added directly to the user dictionary, or a file could be duplicated and configured accordingly. User dictionaries, residing in a dedicated directory, are simple text files with a standard four-line header, containing one word per line. Users can configure one or more files as current user dictionaries.

Finally, let's present post-OCR statistics for a geography book from 1795 [122]. Texts from the 18th century share a language model similar to that of the 17th century, being nevertheless relatively simpler, and partially sharing spelling dictionaries.

The book comprises a total of 5,066 words. In comparison with modern orthography, 613 words (12.1%) exhibit differences. There were 75 OCR errors, with word accuracy exceeding 98.5% and character accuracy surpassing 99.7%.

To assess against modern orthography, our transliteration utility identified 31 erroneous words in the modern script (0.6%). We analyzed these words to refine transliteration, promptly incorporating them into the exception dictionary accompanying the transliteration utility.

Furthermore, 188 words (3.7%) differ in writing, either in a single word or separately, reflecting the historical evolution of orthography.

The remaining 319 words (6.3%) warrant philological expertise to comprehend why their spelling differs. For instance, the word meaning **up to** was printed as both **pânâ** (modern norm) and **pănă**. This discrepancy may result from local or historical variations, or a misprint. While this approach ensures reliable results, it may pose challenges for non-professionals.

Chapter 5

Recognition of texts printed with transitional alphabets

5.1 What is transitional alphabets?

The author of high detailed work “A History of Romanian Spelling” [11] termed transitional Romanian alphabet as the most original graphic systems in the modern history of European cultures. The necessity of transitional alphabet, however, sprang from the common European phenomenon. The majority of the first printed books were the cult books. The originality is based on a combination of two factors specific to the Romanian language. The cult books in Romanian regions are written by Cyrillic scripts. However, the Romanian language belongs to Latin Group and its natural script is Latin. The transition of all Romanian writing to Latin scripts was declared at 1830, but the process was not quick and easy. Thus, transitional alphabets were used in the Romanian typography since 1830 until 1870.

The specific work, that concerned transitional alphabet[123], shows detailed process of migration. This process was so complicated and intricate that the author called it a “saga”. This process was not even straight-forward, being performed in ”trial and error” way. After unsuccessful attempts, the printing house temporarily returned to Cyrillic letters.

The migration process was performed by one of two techniques. The first

technique consists in direct transliteration. This technique succeeded, for example, in transition of specific Cyrillic letters of diphthong: ѣ→ia, ѥ→ie. But direct transliteration was rejected as representation of "phonetic" sense. In traditional Latin orthography "phonetic" sense is, usually, represented by several letters, like, for example, in English: sh /ʃ/, ch /tʃ/. But multi letters printing was often not accepted by readers, so direct transliteration technique was rejected. The second technique consists in the interpretation of the transitional alphabet to the extent that the sound meaning of the syllables and the letters was as close as possible. This technique was accepted as the main one. The transition rules have been discussed and tested for a long time. For some sounds, the multi-letter representation was replaced by the design of own, specific Romanian, letters, and as a result, the Romanian diacritics was born.

Such a complex and lengthy transition process leads to difficulties with the formal definition of the transitional alphabet. But the general answer to the question: what are transitional alphabet? - follows. The transitional alphabet is defined as 36 letters, 27 of which are modern ones, plus 9 old letters: Ѣ ѣ, Ѥ ѥ, Ѧ ѧ, Ѩ ѩ, Ѫ ѫ, Ї ї, Ъ ъ, Щ щ, Ц ц.

5.2 Specific aspects of transitional alphabets digitization

The formal content of the transitional alphabet exists mainly for reference, because, actually, Cyrillic and Latin letters were mixed in different proportions depending on the time, place, preferences of the typographer, editor or author of texts.

Book [123] count up to 17 variants of the transitional alphabets, but some other authors declared about 20 variants. These transitional alphabets variants cannot be ranged in any order. Book [123] shows the "highest score" example of 1840 where the title page was printed in four different scripts simultaneously (old Cyrillic, simplified Cyrillic, transitional, and Latin).

From the point of digitization, such an irregular variety of transitional alphabets creates additional problems in both OCR and the visual representation of resulted document.

The visual representation problems can be solved by usage of our specific font[85] as it was described in previous chapter. The maximum number of Cyrillic letters used in different variants of transitional alphabet is 43.

The problems of OCR of transitional alphabet are more complicated. To get acceptable accuracy, OCR tools have to be prepared for a particular variant of transitional alphabet: to be configured, be trained and be supplied by the proper dictionary.

During the development of our platform, two approaches were tested.

The firstly tested approach is to reproduce the resulted text in its original variant of transitional alphabet. AFR, prepared as it was described above, produces 7% of erroneous words. This is a good result, but the preparation process takes a lot of time and resources.

To achieve more effectiveness, the second approach was tested. This approach uses the general feature of large OCR systems, in our case AFR, to output the result both using original glyphs and substituting them by any sequence of letters from the selected alphabet of recognition. This is called ligatures in AFR documentation. So, the second approach consist in using as ligature the Latinized version of the transitional alphabet that we the specifically developed. This intermediate alphabet can be set in one-to-one mapping with any transitional alphabet. For example, both **т**(Cyrillic) and **t** (Latin) will be recognized as **t**. Because of one-to-one letter mapping, the utility added to the our package converts the OCR output into the desired variant of the transitional alphabet.

The second approach proved fruitful. The OCR errors reduced to 4.8%. This approach also reduces the volume of the dictionary. For example, **trekut** (modern Latin script **trecut**) in the recognition dictionary may check up to **16 variants** obtaining by independently replacing **t**→**т, r**→ **p, k**→**к, u**→ ꙋ.

The second approach as well solves technical problem of OCR engine setting. AFR, for example, does not support arbitrary Unicode glyphs in its dialogs and forms. Old Romanian letter ѧ was introduced in Unicode only after 2009. Standard system fonts do not contain some Romanian Cyrillic letters. As a result, we see in AFR empty boxes instead of letters during training, alphabet formation, etc. The use of ligatures allows to employ fonts only at the stage of converting the output data, when we control the view.

Another problem of OCR of transitional alphabet printings arose during OCR engine training. There are very few of the existing historical lexicons for results validation. Actually, we had to create all required resources from scratch. We repeated recognition several times using the results as sources for the addition to both dictionary and glyph patterns library.

Resuming, the OCR of texts printed by Romanian transitional alphabet differs from described above OCR of Romanian Cyrillic alphabet by more complex reconstruction procedure. Depending on particular task, user can use or does not use the utility for full reconstruction of original text in digital form.

5.3 OCR of Romanian transitional alphabet by examples

In this section we demonstrate the variations of Romanian transitional alphabet depending on time and region. For the demonstration we extracted from [11] the list of examples which have some particular and interesting features. The scanned sources according this list were obtained from free web bases.

5.3.1 Initial usage

The arrival of a transitional alphabet, both in Muntenia and Moldova, dates back to 1829-1830. One of the most known examples of the initial transitional alphabet usage is Iasi newspaper *Albina românească.* Initially, the appearance of Latin letters was very rare, as we can see in an example taken from[124] on Fig. 5.1, which is a fragment of the first issue on June 1, 1829. The text is practically entirely printed in Cyrillic script, except for a single Latin letter **i**. The low quality of the original text has been improved by image pre-processing; however, two recognition errors remain.

In later issues, especially in the literary supplement *Alăuta Românească* [124], Latin characters become more and more frequent (Fig. 5.2). One can see that the title is printed entirely in Latin letters and the following text is in Cyrillic with full replacement of the Cyrillic **и** with the Latin **i**. For the spelling of the letter **t**, two variants are used: **т** and **ҭ**. The word **кѵрс** is

АЛБИНА РОМѪНѢСКЪ

ГАЗЕТЪ ПОЛИТИКО-ЛИТЕРАЛЪ.

Ешїи 1 юнїе 1829.

АЛБИНА РОМѪНѢСКЪ

ГАЗЕТЪ ПОЛИТИКО-ЛИТЕРАЛЪ.

ѪНАИНТЕ КѸВѪНТАРЕ.

Епоχа л карѣ тръим поартъ семне ѫс8шите ши вредниче де мираре! Дор8л лвъцът8рилор н8н8май къ лфръщеще пе лък8иторїй оүнеи цери лтр8 кѫщигарѣ ачестеи моралниче ав8ціи, прин карѣ ѡ нацїе се фаче п8тертикъ ши феричитъ, чи лкъ ши ѡамени не асемънаци к8 Релігїа, к8 лимба ши к8 леңиле сѫнт

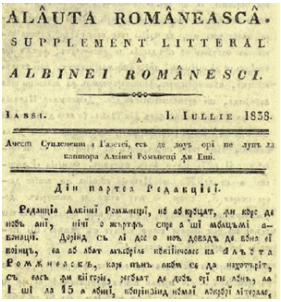
Въцйнд8се ши с8п8инд8се дрептелор леңй. Оаре п8тем8 нои приви ла ачѣсте б8не оүрмате лнаинтѣ ѡкилор ностри, фъръ а ни лчриста къ н8май нацїа ноастръ лчѣ маи маре парте есте лиѱитъ де ачести лб8нътъцирй ши лнапоетъ де кѫт тоате нѣм8риле Европей, ши де кѫт м8лте алтеле че лък8еск8 пре челе лалте пърцй але пъмѫнт8л8й? Чине н8 сѫмте л цара ноастръ лиѱа ашезъмѫнт8рилор

Figure 5.1: The first issue of *Albina românească*, June 1, 1829.

recognized erroneously as **кope**.

5.3.2 Complete usage

The next example presents full version of transitional alphabet how Heliade Radulescu, Balcescu, Treboniu proposed it and Laurian, Asachi, Kogalniceanu, etc. used it. We selected as an example an excerpt from the famous author Vasile Alecsandri's novel *Suvenire din Italia: Buchetiera de la Florența* (*Souvenirs from Italy: A Florist from Florence*) [125, 126] (Fig. 5.3) because the works of the classical authors were re-published by modern Romanian alphabet. Modern re-publications are a useful source for verifying the transliteration tool. The transliteration tool, in turn, can be used for creation of datasets for both training and validation of the OCR engine. The quality of OCR is very good, with only one error in this fragment (the Latin letter **d** recognized as



ALÂUTA ROMÂNEASCÂ.
SUPPLEMENT LITTERAL
A
ALBINEI ROMÂNESCI.
IASSI. 1. IULLIE 1838.

Ачест Суплемент а Газетеі, есъ де доуъ орі пе лунъ ла
бантора Албінеі Ромънещі Лн Еші.
Дін партеа Редакціеї.
Редакціа Албінеі Ромжнещї, н8 а8 кр8цат, Лн коре де но8ъ анї, нічї о жъртфъ спре а 'шї м8лцъмі абонацій. Дорінд съ лі дее о ноъ довадъ де в8на еї воінцъ, ел а8 л8ат мъс8ріле к8вїінчоасе ка Алъ8та Ромжнеаскъ, каре пънъ ак8м се да нехотъріт, съ еасъ Лн віігоріе, рег8лат де до8ъ орї пе л8нъ, ла 1 шї ла 15 а л8неї, к8прінзінд н8маї л8кр8рї лїтерале прек8м ачест Лнтъї н8мър. Редакціа се ва сірг8і ка Алъ8та съ ръс8не пр8д8кт8ріле д8χ8л8ї челе маї нсъ шї челе маї інтересанте пентр8 четіторї. Еа Лшї ва Лмпліні скопос8л к8 аж8тор8л м8лтор тінерї літерарї, карії а8 БІНЕВоіт а фагъд8і л8кръріле лор ачестеї п8блікацій періодіче.

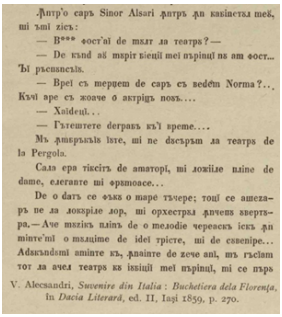
Figure 5.2: *Alăuta Românească*, July 1, 1838.

the Cyrillic letter **б**). Most of the letters are Cyrillic, with the letters **D**, **d**, **i**, **m**, **n** being written in Latin script. In the case of the last letter, the Cyrillic spelling **н** is also found. Note that all proper names: **Sinor Alsari**, **Norma**, and **Pergola** are written in Latin characters.

Another example of using the transitional alphabet in its mature state is interesting by the confident use of the transitional alphabet in secular literature. The example is from the novel *Radu VII from Afumați* [127] by Stefan Andronic (Fig. 5.4A, p. 91). The *Dictionary of Romanian literature* declares this novel as the first Romanian historical roman.

5.3.3 One direction conversion Cyrillic – Transitional — Latin

The book[11] presents the examples which are named *effective transition*. The definition of *effective* here means that the mentioned printed publications: started issuing using the transitional alphabet immediately after the



Ѧнтр'о саръ Sinor AIsari Ѧнтръ Ѧn кабінет8л ме8, ші ъмї зісъ:
— В*** фост'аї de m8лт ла театр8? —
— De кънd a8 m8ріт біеції меї пърінцї н8 am ФОСТ... Ъї ръсп8нсъї8.
— Вреї съ мерцет de саръ съ bedem Norma?... Къчї вре съ жоаче о актріцъ по8ъ...
— Хаїдецї...
— Гътештете деграбъ къї време...
Мъ Ѧмбръкъї8 ї8те, ші не d8съръм ла театр8 де In Pergola.
Сала ера тіксітъ de аматорі, ші ложііле пline de dame, елегапте ші фр8moасе...
De o датъ се фък8 o маре тъчере; тоцї се ашезаръ пе ла лок8ріле лор, ші орхестр8л Ѧпчеп8 8верт8ра, —Аче m8зікъ пліпъ de o мелodie череаскъ іскъ Ѧn minte'mї o m8лціme de ideї трісте, ші de с8вепіре... A68кънd8mї aminte къ, Ѧпainte de зече ani, мъ гъсїam тот ла ачел театр8 к8 і8біції меї пърінцї, ші се пър8

Ѧнтр'о саръ Sinor AIsari Ѧнтръ Ѧn кабінет8л ме8, ші ъмї зісъ:
— В*** фост'аї de m8лт ла театр8? —
— De кънd a8 m8ріт біеції меї пърінцї н8 am ФОСТ... Ъї ръсп8нсъї8.
— Вреї съ мерцет de саръ съ bedem Norma?... Къчї вре съ жоаче о актріцъ по8ъ...
— Хаїдецї...
— Гътештете деграбъ къї време...
Мъ Ѧмбръкъї8 ї8те, ші не d8съръм ла театр8 де In Pergola.
Сала ера тіксітъ de аматорі, ші ложііле пline de dame, елегапте ші фр8moасе...
De o датъ се фък8 o маре тъчере; тоцї се ашезаръ пе ла лок8ріле лор, ші орхестр8л Ѧпчеп8 8верт8ра, —Аче m8зікъ пліпъ de o мелodie череаскъ іскъ Ѧn minte'mї o m8лціme de ideї трісте, ші de с8вепіре... A68кънd8mї aminte къ, Ѧпainte de зече ani, мъ гъсїam тот ла ачел театр8 к8 і8біції меї пърінцї, ші се пър8

Figure 5.3: Fragment of *Souvenirs from Italy: A Florist from Florence* by V. Alecsandri, 1840

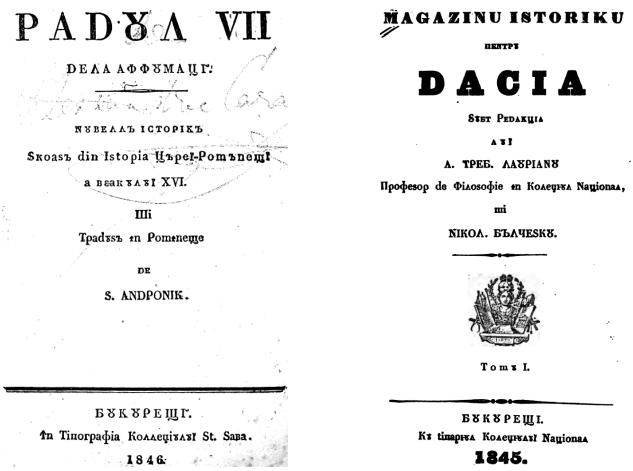
announcement; used the transitional alphabet only for a few years; finally, were issued entirely in the Latin alphabet without any returns.

The presented example is the fragment from *Magazinu istoricu pentru Dacia*[128] (Fig. 5.4B, p. 91), edited by Laurian and Balcescu, that used the transitional alphabet in 1845—1847 and then was issued by pure Romanian Latin alphabet.

5.4 Accuracy evaluation

Here we propose analysis of accuracy in recognition of p. 20 of the novel *Radu VII from Afumați* (see Sec. 5.3.2 above).

The most frequent errors were observed in the letter **б** (changed to **в**) and the letter **п** (changed to **п**). With total 1172 characters on the page and 93 erroneous characters, we have the 92.1% accuracy.



A. B.

Figure 5.4: A. Cover of *Radu VII from Afumați* by S. Andronic, 1846
B. Title page of *History Magazin for Dacia*, 1845

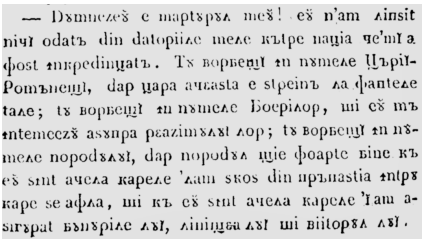
This result could be improved by adding a user dictionary as part of the FineReader language model. With the dictionary, we got only 31 erroneous characters, and the accuracy became 97.4% that seems to be a good result.

However, after adding the dictionary, while most errors with the letter **n** were resolved (as similar words were added to the dictionary), there were still recognition errors with the letters **ш**, **б**, **m**, and some others. Also, the word **іубеще** was incorrectly recognized this second time, even though the model recognized it correctly without the dictionary.

The model almost flawlessly recognizes special characters and punctuation, often erring in the letters **n**, **п**, **ш**, **m**, **б**, and **ц**. The model learned to recognize the letter **ѵ** well and also correctly recognizes the letter **л**.

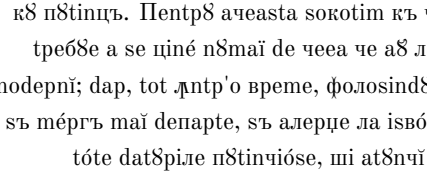
The letter **ε** was not on this particular page, but it was often identified as **e** in other pages, although in reality, this letter later transitioned into the **e**.

Later it was observed that the letter **î** also appears in texts but was always replaced with **i**. The letter which looks like **ѵ** with a line over it was counted as a simple **ѵ**.



-- Дѫмнезеѫ е мартѫрѫл меѫ ! еѫ нам ліпсит нічї одатъ дін даторііле мелє кътре націа чемь а фост ѫнкредінцатъ. Тѫ ворбещї ѫн нѫмеле Църїй-Ромънещї, дар цара ачеаста е стреінъ ла фаптеле тале, тѫ ворбещї ѫн нѫмеле Боерілор, ші сѫ мъ ѫнтемеезѫ асѫпра реазімѫлѫї лор; тѫ ворбещї ѫн нѫмеле нородѫлѫї, дар нородѫл щіе фоарте біне къ еѫ сѫнт ачела карєле 'ѣам скос дін пръпастіа ѫнтрѫ каре се афла, ші къ еѫ сѫнт ачела карєле 'ѣам асігѫрат бѫнѫріле лѫї, лініщеа лѫї ші вііторѫл лѫї.

Figure 5.5: Digitization of *Radu VII from Afumati* fragment



Ної кредем къ о асémінє Istopie este кѫ пѫtinцъ. Пентрѫ ачеаста сокotim къ чеї че se окѫпъ кѫ Istopia нóстръ, нѫ требѫe a se цінé нѫmaї de чеea че aѫ лѫкрат ші aѫ zis istopічїї nostрїї чеї modepнї; dap, tot ѫntр'o вреme, фолosindѫse de adевърѫрї deѕкопepite de дъпшїї съ мéргъ маї dепapte, съ алерце ла iѕвóръле opiцiпале, съ кaѫte ші съ adѫne тóte datѫріле пѫtinчióse, ші atѫnчї вор пѫté цese o Бѫнъ Istopie.

Figure 5.6: Digitization of *Magazinu istoricu pentru Dacia* fragment

5.5 Final remarks

Digitization of texts printed in transitional alphabets is an important element of digitization of Romanian historical printed publications. The period of using transitional alphabets coincided with the era when printed publications

became a necessary part of everyday life. Thus, this period gave a rich legacy of both periodicals and literature, which can be called the first purely Romanian. So, digitized copies of transitional alphabets prints are interesting not only for linguists but also for historic and literature researchers.

Chapter 6

Transliteration

6.1 Introduction

Transliteration is a type of conversion of a text from one alphabet to another that involves changing letters in predictable ways, such as for Cyrillic д→**d**, Greek χ →**ch/h**, **Latin** æ→ **ae**, or old Romanian Cyrillic ѧ → **î/în/îm** depending on the context [31, 129]. Therefore, transliteration consists of representing the characters of a given alphabet X through the characters of another alphabet Y, maintaining (as much as possible) the reversible operation.

Up to now, several transliteration techniques between two scripts have been proposed. Particular interest has been aroused by works focused on the orthographic transliteration of English proper names into Chinese, Japanese, Korean, or Arabic. In 1997, a method of transliteration between Japanese and English was introduced, using translation algorithms based on finite state machines, and this method was adapted the following year for bidirectional transliteration between English and Arabic [130].

In [131], a transliteration technique called Direct Orthographic Mapping (DOM), or in other words, an n-gram based transliteration model, is proposed. The DOM approach attempts to model the phonetic equivalent association by fully exploring orthographic contextual information and orthographic mapping. In the DOM technique, a common transliteration model is pre-

sented to capture the source-target orthographic mapping relationship and contextual information. The proposed framework is applicable to all pairs of foreign languages.

The transliteration of Romanian texts from the „Moldovan" script to the Latin script in our country began in 1989 with the application of Law No. 3462 of August 31, 1989, adopted by the Parliament of the Republic of Moldova [132]. The standards of transliteration to English can not be adopted directly, because of Romanian Latin script specific features. For example, in the tradition of the Romanian language, the Cyrillic letter **х** is transliterated by the Latin letter **h**, but in the tradition of English-speaking countries, the same letter is transliterated as **kh**. Thus, in Romanian, the token **Сахалин** is transliterated as **Sahalin**, while in English it is **Sakhalin** [133].

6.2 Transliteration Rules for Romanian Cyrillic

The presentation of digitized Romanian Cyrillic documents was described in previous chapters: for Romanian Cyrillic (RC) in Chapter 4, for Transitional Romanian (TR) in Chapter 5. Summarizing, the digital Romanian Cyrillic letters are mapped by their Unicode values and presented by special fonts.

The representation of Moldavian Cyrillic (MC) poses no problems, because the only difference with the Russian is letter **ӂ** that is presented in commonly used fonts.

The correspondence of specific letters from the RC with the modern Romanian Language (MRL) and their mapping into Unicode is presented in Table 6.1, p. 96.

Some accented or combined letters are meanwhile missing and should be specially treated, for example, **ў (ŭ)** or **і- ў (iŭ)** in TR. To present them in Unicode, it is necessary to use combining accents, and we can't fully reproduce subtle details of the graphical presentation of the original text.

6.3 MC: Bidirectional Transliteration

The transliteration MC⟶MRL was discussed in detail in [134]. There are three groups of rules. Most letters (26 of 31) can be mapped directly as shown

Table 6.1: Correspondence of some specific letters from the RC with MRL

RC	MRL	The Unicode code for the letters from the RC
“ Ѣ”	“Ea”	0462
“ ѣ”	“ea”	0463
“ Ꙗ”	“Ia”	0465
“ ꙗ”	“ia”	0464
“ Ѧ”	“Î”, “În”, “Îm”	A64E
“ ѧ”	“î”, “în”, “îm”	A65F
“ Ѹ”	“U”	A64A
“ ѹ”	“u”	A64B
“ Ҁ”	“C”, “Ch” (before **e, i**)	041A
“ ҁ”	“c”, “ch” (before **e, i**)	043A
“ Ӑ”	“Ă”	042A
“ ӑ”	“ă”	044A
“ Ꙉ”	“Șt”	0429

in Table 6.2, p. 96.

Table 6.2: MC→MRL: direct letter mapping

MC→MRL	MC→MRL	MC→MRL	MC→MRL	MC→MRL	MC→MRL	MC→MRL	MC→MRL
а→a	б→b	в→v	д→d	е→e	ж→j	ӂ→g	з→z
и→i	й→i	л→l	м→m	н→n	о→o	п→p	р→r
с→s	т→t	у→u	ф→f	х→h	ц→ț	ш→ș	ь→i
э→ă	ю→iu						

Context rules exist for three letters as shown in Table 6.3, p. 97.

The letter **ы→â, î,** where **î** is written at the beginning or end of words, while **â** inside words. The difficulty is that **î** is kept after prefix, for example, **ne+însoțit** = **neînsoțit** (unaccompanied).

The letter **я→ea, ia**, a presents the biggest problem that can't be fully solved without access to dictionaries. Rules are mostly heuristic and statisti-

Table 6.3: MC→MRL: Context Rules in the Order of Application

MC	MRL	Context
г	gh	before **е, и, ь, ю, я**
г	g	otherwise
кс	x	as exception, examples: **eczema** and derivatives, **Alecsandri**
к	k	as exception, examples: **kilogram**, **Kogălniceanu**, etc.
к	ch	before **е, и, ь, ю, я**
к	c	otherwise
ч	c	before **е, и, ь, я**
ч	ce	before **а**
ч	ci	otherwise

cal, and over **20 rules** do not cover all cases. This situation exists because MC was not thoroughly designed but is an irregular mapping of Romanian sounds to the Russian letters.

Some words can't be transliterated according to these rules: foreign proper nouns and words of foreign origin that keep their writing in MRL. We use the exception dictionary for them.

The inverse transliteration MRL→MC (1967–1989) was mainly necessary to produce a word list in MC from the existing word list in MRL. This task equally meets difficulties, mainly with the letter i. In particular, at the word ends **i** may be omitted, or converted to **и, й, ь**. Examples: **arici→арич** (hedgehog, singular), **arici→аричь** (hedgehogs, plural), **[a] cheltui→[а] келтуи** ([to] count; stress on **i**), **[eu] cheltui→[еу] келтуй** (I count: stress on **u**). Analogous problems appear at the transliteration of diphthongs and triphthongs. For example, diphthong **ia→я, ия, иа**: **soia→соя** (soybean), **caucazian→кауказиян** (Caucasian), **cartezian→картезиан** (Cartesian).

Some of these problems could be solved by consulting Morpho-Syntactic Data (MSD), which was proposed in the framework of the project MULTEXT-East [135]. In the remaining cases, context analysis or even manual intervention could be performed.

The whole transliteration process is implemented as a set of filters each modeling a separate situation. The filters are:

- prefix filters;
- suffix filters;
- diphthong and triphthong filters;
- final filters (letter→letter).

Prefix filters are created separately for words that begin with the same letters (**creast*→кряст***, **crea*→креа***, **paie*→пае***, etc.). At transliteration, these filters are applied at first.

Suffix filters are common for all words in the lexicon. They can be divided into two classes: conditional depending on the MSD value, and unconditional.

Diphthong and triphthong filters aim to transliterate some letter combinations like **ie**, **io**, **eio**, **chio**, etc. They are applied independently of position and context.

Final filters transliterate all letters that remain after the application of other filters. For example, **d→д**, **c→к**, **ş→ш**.

Some filters can use results from the previous filters. Such filters may look like: **celуй→челуй, ieпь→ень**, combining Latin and Cyrillic letters.

If the situation is ambiguous, and expert's intervention (manual selection) is necessary, alternatives can be generated, for example **кафен[иул][юл]** with the result **→кафениул** (brownish, coffee color; with the definite article); **ча[иул][юл]→чаюл** (the tea; with the definite article).

This algorithm was applied to the lexicon elaborated at the Al. I. Cuza University in Iași [136]. The automation rate of transliteration was approx. 90%.

6.4 Transitional Alphabets

In chapter 5 the problem of multiple (more than 17) variants of TR alphabet is discussed in details. For initial transliteration tool performing the starting variant had to be selected.

The transliteration tool of HeDy deals with the variant from formal definition: 36 letters, 27 of which are modern ones, plus 9 old letters: Ѣ ѣ, Ѥ ѥ, Ѧ ѧ, Ѩ ѩ, Ѵ ѵ, І і, Ѯ ѯ, Ѱ ѱ, Ѳ ѳ.

In the case when a different variant of the TR alphabet is used in the digitized document, the basic algorithm of the HeDy transliteration tool allows you to simply add new letters.

The TR transliteration rules are more simple then ones for MC, because TR has not dubious rules.

Transliteration of 32 letters is performed under direct rules (Table 6.4, p. 99).

Table 6.4: TR⟶MRL: direct letter mapping

MC→MRL	MC→MRL	MC→MRL	MC→MRL	MC→MRL	MC→MRL	MC→MRL	MC→MRL
а⟶a	б⟶b	в⟶v	д⟶d	е⟶e	ж⟶j	з⟶z	и⟶i
й⟶i	л⟶l	м⟶m	н⟶n	о⟶o	п⟶p	р⟶r	с⟶s
т⟶t	ф⟶f	х⟶h	ц⟶ţ	ш⟶ş	щ⟶şt	ь⟶i*	э⟶ă
ю⟶iu	ѣ⟶ea	ѥ⟶ia	ѧ⟶â	ѹ⟶u*	ъ⟶i*	ѫ⟶ă	ѭ⟶g

* At linguists' request, rules **й,ь,** ĭ⟶ ĭ, and ѹ⟶**ŭ** may be applied.
** Before **е, і** only.

The four remaining letters are transliterated under context rules (Table 6.5, p. 99).

Table 6.5: TR⟶MRL: Context Rules in the Order of Application

TR	MRL	Context
г	gh	before **е, и, ю**
г	g	otherwise
кѵ	x	exceptions: Table 6.3, p. 97
к	ch	before **е, и, ю**
к	c	otherwise
ч	c	before **е, и**
ч	ce	before **а**
ч	ci	otherwise
ѧ	î*	before **m, n**
ѧ	îm	before **b, p**
ѧ	în	otherwise

* In some texts, always ѧ⟶**î** (simple rule).

6.5 Glyphs and Transliteration Rules for RC

As it was declared in Chapter 4, RC contains 43 letters: Аа Бб Вв Гг Дд Ѥѥ Жж Ѕѕ Зз Ии Її Кк Лл Мм Нн Оо Пп Рр Сс Тт Ѹѹ Ѳѳ Фф Хх Ѡѡ Цц Чч Шш Щщ Ъъ Ыы Ьь Ѣѣ Жж Юю Ꙗꙗ Ѧѧ Ѫѫ Ѭѭ Ѯѯ Ѱѱ Ѵѵ Ѷѷ Ѿѿ.

Glyphs like й and ӳ weren't treated as separate letters in RC, but as и and ӯ with diacritic signs.

37 letters are transliterated under direct rules (Table 6.6, p. 100).

Table 6.6: RC→MRL: direct letter mapping

RC→MRL	RC→MRL	RC→MRL	RC→MRL	RC→MRL	RC→MRL	RC→MRL	RC→MRL
а→a	б→b	в→v	д→d	е→e	ж→j	ѕ→dz	з→z
и→i	ї→i	л→l	м→m	н→n	о→o	п→p	р→r
с→s	т→t	ѹ→u	ѹ→u	ф→f	х→h	ѡ→o	ц→ţ
ш→ş	щ→şt	ъ→ă	ы→î	ь→i	ѫ→â	ю→iu	ꙗ→ia
ѳ→t	ѱ→ps	ѯ→x	ѵ→i	ѷ→g			

* Before **е, і** only.

The remaining 6 letters need context rules (Table 6.7, p. 100).

Table 6.7: RC→MRL: Context Rules in the Order of Application

RC	MRL	Context	RC	MRL	Context
г	gh	before **е, и, ї, ю**	ѧ	a	at the beginning of word
г	g	otherwise	ѧ	a	after **ї, ц**
кс	x		ѧ	e	after **ч**
к	ch	before **е, и, ї, ю**	ѧ	ea	after another consonant
к	c	otherwise	ѧ	ea	at the end of word
ч	c	before **е, и, ѣ**	ѧ	ia	otherwise
ч	ce	before **а**	ѫ	îm	before **b, p**
ч	ci	otherwise	ѫ	în	otherwise
ѣ	e	after **ч** (exception **ч ѣ→cea**)			
ѣ	ea	otherwise			

6.6 Examples of Transliteration

This section provides examples of transliteration that demonstrate how useful this HeDy tool is for historians and linguists, especially for those who are not familiar with Cyrillic writing.

The example in Table 6.8, p. 101 is the text from the "New Testament" book printed in 1648, in Balgrad (Alba-Iulia) (described in Chaper 4).

Table 6.8: Example of 17 century document transliteration

Text after OCR	**Transliterated text**
Де ꙟчепȣт Єра кȣвънтȣл, ши кȣвънтȣл ачела Єра ла дȣмне□ꙁъȣ, ши дȣмне□ꙁъȣ Єра кȣвънтȣл ачела. Ачеста кȣвънт Єра деꙟчепȣть ла дȣмне□ꙁъȣ. Тоате пре ꙞЄль фȣрѫ фъкȣте; ши фърѫ Єль немикѫ нȣ фȣ фъкȣть варе че е фъкȣть. ꙟтрȣ Єл Єра вїѩца ши вїѩца Єра лȣминѫ ѡаменилѡр. Ши лȣмина ачѣѩ Ꙟтрȣ ꙟтȣнѣрекь стрълȣчїѩ ши ꙟтȣнѣрекȣль преѩ нȣѡ кȣпринсе.	De început Era cuvântul, și cuvântul acela Era la dumnezău, și dumnezău Era cuvântul acela. Acesta cuvânt Era deînceput la dumnezău. Toate pre ÎNEl fura făcute; şi făra El nemica nu fu făcut vare ce e făcut. îtru El Era viața şi viața Era lumina oamenilor. Și lumina aceaia Întru întunearec strălucia şi întunearecul preia nuo cuprinse.

The example in Table 6.9, p. 102 is the text from the novel "Radu VII from Afumati"(1846) by Stefan Andronic (described in Chapter 5).

6.7 Transliteration utility

This section introduces a tool designed for transliteration from RC, TR, and MC to MRL.

The essence of this tool lies in its algorithmic approach to transliteration. It functions based on a system of parameterized rules applied to each char-

Table 6.9: Example of transitional alphabet document transliteration

Text after OCR	Transliterated text
Дръган зикꙟнд ачесте ворбе, се лъсасе а се стъпꙟнї де патима са, де ꙋръ, шї де ресбꙋнаре; фокꙋл мънїеї ї се ꙋркасе ꙟн ображї; џестеле сале ера смѧчїтеѣ, кꙋ вїоїчїꙋне мїшкате; кътꙋтътꙋра са ꙟшї ре- лꙋасе ꙟндръзнеала, ачеа ꙟндръзнеалъ ꙟнсъ че карактерїзеазъ пе омꙋл че се сїмте слаб, шї кареле креде къшї а редобꙟндїт вреднїчїа са. Радꙋ, дїн потрївъ, ера лїнїщїт, шї, аскꙋлтꙟнд їнсꙋлтеле че протївнїкꙋл съꙋ ꙟї адреса, нꙋ дете нїчї ꙋн семн де тꙋлбꙋраре, нїчї де мꙟнїе, дар ръспꙋнсе: - Дꙋмнезеꙋ е мартꙋрꙋл меꙋ ! еꙋ нам лїпсит нїчї одатъ дїн даторїїле меле кътре нацїа чемь а фост ꙟнкредїнцатъ. Тꙋ ворбещї ꙟн нꙋмеле Църїї-Ромънещї, дар цара ачеаста е стреїнъ ла фаптеле тале, тꙋ ворбещї ꙟн нꙋмеле Боерїлор, шї сꙋ мъ ꙟнтемеезꙋ асꙋпра реазїмꙋлꙋї лор; тꙋ ворбещї ꙟн нꙋмеле народꙋлꙋї, дар народꙋл щїе фоарте бїне къ еꙋ сꙟнт ачела кареле лам скос дїн пръпастїа ꙟнтрꙋ каре се афла,шї къ еꙋ ачела кареле Їам асїгꙋрат бꙋнꙋрїле лꙋї, лїнїщеа лꙋї шї вїїторꙋл лꙋї.	Drăgan zicînd aceste vorbe, se lăsase a se stăpîni de patima sa, de ură, şi de resbunare; focul măniei i se urcase în obraji; jestele sale era smiacitea, cu vioiciune mişcate; căutătura sa îşi re- luase îndrăzneala, acea îndrăzneală însă ce caracterizează pe omul ce se simte slab, şi carele crede căşi a redobîndit vrednicia sa. Radu, din potrivă, era liniştit, şi, ascultînd insultele ce protivnicul său îi adresa, nu dete nici un semn de tulburare, nici de mînie, dar răspunse: - Dumnezeu e marturul meu ! eu nam lipsit nici odată din datoriile mele către nația cem a fost încredințată. Tu vorbești în numele Țării-Romănești, dar țara aceasta e streină la faptele tale, tu vorbești în numele Boerilor, şi su mă întemeezu asupra reazimului lor; tu vorbești în numele norodului, dar norodul știe foarte bine că eu sînt acela carele lam scos din prăpastia întru care se afla,şi că eu acela carele Iam asigurat bunurile lui, liniștea lui şi viitorul lui.

acter of a word in the source alphabet (RCA). The algorithm translates each character x_n, to its corresponding character y_n, in the target alphabet (MRA), forming a sequence that replicates the transliterated form of the word. The result represents a sequence of characters whose concatenation reproduces the transliterated form of the word Y. In the given context, we have the following definitions for the variables in the formula: $y_n = \text{Trans}\,(x_n, \text{Pos}(n, X))$

1. y_n: the n-th character of the transliterated word Y.
2. x_n: the n-th character of the transliterated word X.
3. X: the original word in the source writing system.
4. Y: the transliterated word in the target writing system.
5. n: the current position of the character in the word (starting from 0).
6. *Trans()*: a function that applies the transliteration rules for a given character (x_n) and its position (n) in the word X.
7. *Pos(n, X)*: a function that determines the position of the character x_n in the word X.

Despite its structured approach, the tool accounts for exceptions like foreign words, proper names, etc., employing specialized methods, such as dictionaries, for these cases.

The back-end part of the application is written in Java, using technologies compatible with all Unicode characters. If the font is registered and installed in the operating system, the Java methods used for the interface solve the issue of displaying characters with a simple reference to the specific font.

The first graphical interface of the transliteration utility is built using *JavaFX* and has a design suitable for a desktop application. The application is compatible with the following file formats: *.doc, .docx, .rtf, .txt.*

Considering the requirement to ensure broad access to this service, a web version of the transliteration tool has also been developed, called ***AAConv1***, which consists only of the front-end part, connected to the previously developed back-end. The technologies used for developing the web application are the usual ones, such as HTML, CSS, and JavaScript.

The desktop application is shown in Fig. 6.1 on p. 104.

Figure Fig. 6.2 on p. 104 illustrates the corresponding web application.

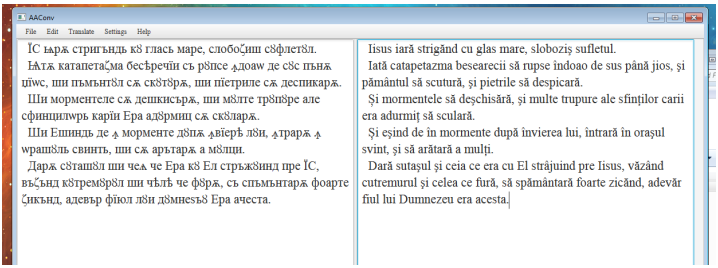


Figure 6.1: The Desktop Transliteration Application **AAconv**

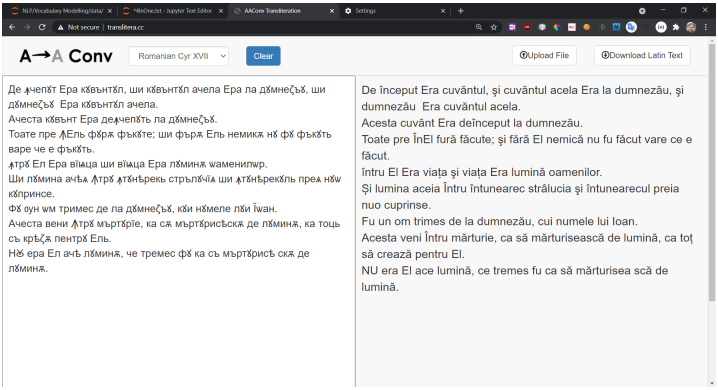


Figure 6.2: The Desktop Transliteration Application **AAconv**

6.8 Comparative Analysis of the Transliteration Process for Cyrillic Script of Different Periods

In this section, we present a comparative analysis of the transliteration process for historical Romanian Cyrillic scripts of different periods.

The best result is obtained for RC, the accuracy of conversion being up to 98%. Comparing transliteration of 1830–1860 and 1945–1989 Cyrillic scripts we will mention the following important aspects. For letters that are identical in both scripting and are transliterated applying elementary rules, the process is the same. There are some letters (г, к ч, ц) the transliteration rules for which are not so elementary but are also identical for any Cyrillic scripting.

Transliteration of 1830–1860 Cyrillic script gives, nevertheless, better results than processing of 1945–1989 Cyrillic script.

Transliteration of transitional alphabets was successful for 96% of words while for MC this fraction was 95%. TR of 1830–1860 has no problems with letters **ы** and **я**. The rule for **ы** that should be converted to **â** or **î** has some fuzziness, namely, keeping **î** after prefixes. Transliteration of **я** from MC creates several ambiguous situations and strongly depends on the context. The most complicated case is the occurrence of **я** inside words. Three variants are **я→ea**, **я→ia**, **я→a**. We use some heuristically and statistically motivated rules but most cases imply addressing external dictionaries. In 1830–1860 TR did not provoke such issues because the letter **я** was not used in every phonetically suitable situation. The same situation is attested in RC, which contains specific letters, for example, **ѣ→ea**; **ѥ→ia**.

Chapter 7

Heterogeneous documents processing

7.1 Definition of heterogeneous content

In most documents, whether they are old or contemporary, you can find structured elements beyond the text itself and images. A good example is an encyclopedia, which usually contains not only text but also various content types like mathematical and chemical formulas, musical scores, diagrams, technical drawings, chess notations, electronic circuits, etc. When digitizing such documents, it's essential not only recognize the text but also handle these diverse components.

Heterogeneous content is associatedt with the possibility to present it in a scripting language [8]. It exhibits the following characteristics:

- it's not exclusively in natural language;
- it involves one or more scripting languages than can present its components;
- the graphical representation can be regenerated using these scripts.

The rapid development of digitization textual documents has resulted in numerous robust and efficient solutions. Researchers often leverage these methods to address the broader challenge of digitizing heterogeneous content. However, handling heterogeneous content poses unique challenges that

cannot be resolved solely by adapting methods for processing pure textual scans. Broadly, many of these approaches use Deep Learning with additional techniques, such as dynamic programming [137] or SVM [138]. A comprehensive review of 29 script text mining techniques, along with accuracy assessments, is available in [139].

Recent research has introduced comprehensive frameworks for digitizing heterogeneous content [138, 140–143]. These frameworks vary in architecture, serving either for whole-archive digitization or partial processing on request. Despite their differences, the architecture and workflow of these frameworks share common patterns. Online services can either be integrated into the overarching framework or establish their independent online systems [144, 145]. The set of services required by domain researchers can also be well-defined.

Recognizing documents with heterogeneous content is a complex task, necessitating a thorough analysis of the image and its segmentation into homogeneous segments. The development of digitizing standard scanned text documents has yielded numerous reliable solutions to the problem of recognizing homogeneous content. However, addressing the specific challenges of heterogeneous content requires more than just adapting methods designed for pure text scans. In most cases, these methods involve Deep Learning with additional enhancements.

Therefore, there is a pressing need to establish a framework that can support the digitization of heterogeneous texts, automate processes where possible, and engage with users when manual intervention or expert opinions are required. This will be discussed in the next chapter.

This chapter discusses the problems at the recognition of documents that contain heterogeneous formalized script-presentable content (mathematical and chemical formulas, music scores, etc.). Due to its great diversity, recognition of such content can't be performed by universal software system. Equally, there isn't any uniform script presentation of the recognition results. State of the art in this area is reviewed. The relevant achievements are systematized. Problems, further development directions and possible solutions are identified. Functionality of Web platform for recognition of heterogeneous documents will be discussed in Chapter 8 below.

7.2 Layout analysis

Concerning the digitization of heterogeneous content, the main problem is layout analysis[146]. The complex page physical layout is converted into logical structure according type of content. Due to the high accuracy of modern OCR systems, today the majority of implementations of heterogeneous documents images digitization don't include the layout segmentation. The engine of any OCR system performs layout segmentation when recognizing plain text document images. Having non plain text regions marked, the process of digitization of heterogeneous content consists in classification, recognition and scripting[147]. The classifier of elements in heterogeneous documents was proposed by [148].

Initially the research was performed to automate classification of heterogeneous content [149], [150], [151]. However, today the classification stage in most works has been separated from the workflow. The main reason for the separation is that the digitization of heterogeneous content in general cases is a time and resources consuming process. Another reason for such separation is that heterogeneous document image classification problem arises mostly once in the process of document archive digitization. After the digital archive is created, new documents are added through an interface, where the type of heterogeneous content is set. The one-time nature of heterogeneous document image classification problem makes it very suitable for online platform processing. Today the set of such platforms includes such giant commercial systems as Cannon IRIS [110] and Kofax Layout Classifier [111]. Beside the commercial systems, many particular projects, especially research ones, develop their own platforms and systems, mostly based on Deep Learning [112]. Some of particular projects concern general classification cases, but their majority consider some particular type of heterogeneous content.

Works on general case classification predominantly employ Deep Learning [114], [115]. An interesting study [116] addresses the processing of heterogeneous document images distorted during scanning. Another unique approach is presented in [117], introducing a hybrid cross-modal methodology that learns simultaneously from a text corpus and image structural information. An alternative to Deep Learning is proposed in [118], where the authors suggest a novel technique for document classification based on

Formal Concept Analysis.

For cases focusing on classification rather than layout analysis, the processing of images with strict layout patterns is notable. An illustrative example is the handling of legal heterogeneous document images [119], where each image must be classified as a specific legal document, with content extraction considering the corresponding pattern. Another example is the standard layout case of newspaper digitization [140], where text and illustrations adhere to strict, usually rectangular, shapes. Additionally, the processing of text type layout elements offers interesting applications, such as identifying specific text areas on identity cards [120].

To mitigate the expenses of general case classification, researchers often confine themselves to a specific set of heterogeneous elements. Numerous content types, with well-developed digitization due to widespread usage, have tools ranging from commercial systems to free online platforms and open-source custom solutions. Noteworthy among them is the **table** content type, where images of simple tables are routinely recognized alongside plain text by modern OCR systems. Although table recognition tools are now included in software like MS Excel, handling specific or complex tables remains an area in development. Current table detection research relies on pre-trained models [152], while some researchers develop their own models and reusable deep learning elements like transformers [153].

Other well-digitized heterogeneous content types encompass mathematical formulae, music scores, technical drawings, data charts, and chemical structures.

7.3 Specifics of heterogeneous document recognition

Content recognition began with text recognition (optical character recognition, OCR). Currently it's an advanced technology, and we describe here state of the art in this area as a point of reference. See more detailed discussion on OCR in Chapter 3 above.

On OCR software, let us take as an example commercial ABBYY FineReader [21] (AFR), and open source free Tesseract [35] and OCRopus [52].

Currently two basic technologies used in text recognition are character patterns and neural networks. Development of neural networks evolved text recognition from recognition of separate characters to the upper levels, namely, to intellectual character recognition and, further, to intellectual words recognition (recognition of whole phrases and lines in a text at once).

OCRopus and Tesseract permit to switch between these technologies.

In comparison with open-source free OCRopus and Tesseract, AFR has numerous advantages. It proposes to its users a convenient graphical shell from which the entire document recognition cycle is available, starting with scanning. It provides complex image correction (for example, correction of trapezoid distortion), page segmentation with automatic detection of the segment type (text, table, or picture), analysis of the table structure, dehyphenation, manual editing after recognition, etc. The user can to input a new language specifying its alphabet, as well as download an additional user dictionary. In difficult cases, AFR can be trained, and the accumulated character patterns (OCR models) can be downloaded and uploaded in new projects.

OCRopus and Tesseract do not provide a single product with a full recognition cycle, as well as a graphical shell. These are packages of separate programs managed from the command line. Ready models exist only for a small number of languages and fonts; for others, it is necessary to perform training and model building from scratch.

The newest AFR 16 is heavy-weight application. Its Windows issue is restricted to 64 bit Windows 10/11.

AFR 16 recognizes texts in 198 languages in any combination, of which vocabulary support is provided for 53.

There are other text recognition systems, but many of them are no longer supported. This is because the OCR system is complex and time consuming to develop. If the system does not meet the enthusiasm of users or does not stand up to competition, the investment of forces and finances in the development instantly loses its meaning. This happened, for example, with CuneiForm that in the beginning of 2000s successfully competed against AFR but was discontinued since 2008.

Some OCR systems that were updated during 2020—2023 are: AFR, Tesser-

act, Ocropus, CIB OCR [154, online, 4 languages only], OCRSpace [155, online, 24 languages], Infty Reader [156, Math also], ReadIris [157], TopOCR [158].

OCRopus was changed the last time in December 2020. TopOCR site is dated 2021.

The universal recognizer Nebo[159] is also noteworthy. Nebo turns handwritten notes, math and diagrams into professional typeset documents. Nebo is AI powered. Its main usage is the recognition of texts and images handwritten on touchscreens, mostly on smartphones.

7.4 Music

The recognition of musical scores (OMR), both printed and handwritten, is a well-developed area. The platform needs to provide specialized OMR scans of scans of historical documents. We obtained the subject area requirements for these items from our past projects [160, 161]. Example of musical score from [162] is shown in Fig. 7.1.

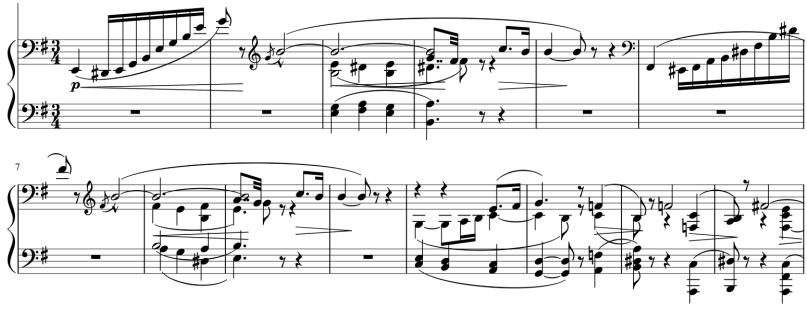


Figure 7.1: Example of musical score (G. Rossini, *William Tell*, overture, solo piano, arr. F. Liszt)

The oldest interpreted record of a religious song (*Hurrian Hymn to Nikkal* [163]) is dated back to 1400 B.C. The method of melody fixing is scripting, and the script even contains metadata. The graphical presentation of music was developed later.

Most music editors use their own file formats. De facto exchange standards between systems are MIDI (Musical Instrument Digital Interface [164])

and the scripting fromat MusicXML [165]. For example, an open source free music editor Musescore [166] provides MusicXML and MIDI data exchange, as well as export and import of some other formats. MSCX, one of its own formats, is also scripting.

MIDI is primarily an industry-standard communication protocol for electronic musical instruments. Information is transmitted in the form of binary messages. Except of music itself, MIDI provides control messages for performance effects, light sources, etc. therefore controlling any attached musical instrument or scenic apparatus.

MusicXML is an XML file with specific tags and attributes for music.

Today the digitization of sheet music both printed and handwritten is well developed domain. There are several free and commercial OMR (Optical Music Recognition) tools. They interpret even manually written scores. The recognition is sometimes less successful on textual parts of music score as these programs are not oriented to OCR.

The list of currently available developments can be found at [167]. The list include both powerful desktop applications for music scores archives creation and online platforms for particular music research. We tested [168], as example, Visiv SharpEye Music Scanning [169] that is one of the most popular OMR platform and has 30-day trial version. SharpEye saves result in MusicXML, NIFF, and MIDI.

Besides full-sized OMR systems, there are a number of small free tools and open source Deep Learning projects for personal purposes, for example, programming of synthesizers. We tested popular solutions like Mozart [170], Orchestra [171], and Audiveris [172].

Commercial score editor Sibelius Ultimate [173] includes *AudioScore Lite* that can input notes by singing or playing a monophonic instrument through a microphone, and *PhotoScore & NotateMe Lite*that turns printed, PDF, JPEG, and even handwrite sheet into editable scores.

Site Musipedia [174] proposes search for music in Internet. The melody can be played from MIDI or virtual keyboard, or whistled in front of microphone. There are also search by contour (graphical pattern formed by sequential notes) and search by rhythm tapped on a specified key.

AnthemScore [175] converts sound (MP3, WAV) to sheet music.

7.5 Mathematics

Standard scripting languages for mathematics are LaTeX [176, 177] and MathML [178].

In general, LaTeX is a huge collection of scripts and scripting tools, including, in particular, nonspecialized vector graphics subsystem MetaPost, and MetaFont to describe fonts. Example of LaTeX script and the resulting formula is shown in Fig. 7.2.

```
$e^x = \sum\limits_{n=0}^\infty \frac{x^n}{n!}$
```

$$e^x = \sum_{n=0}^{\infty} \frac{x^n}{n!}$$

Figure 7.2: Example of LaTeX script and resulting formula

MathML is based on XML and exists in two variants: content MathML, and presentation MathML. The content MathML describes sequence of operations being LISP-like XML implementation of Polish Inverse Notation. This corresponds the newer LaTeX approach to scripting. The presentation MathML directly describes display of formula ignoring its meaning like older TeX.

LaTeX is widely used in the publication of mathematical works and in many other areas. This language is constantly evolving and complemented, in particular in the aspect of the presentation of non-textual elements. For example, in mathematics, very complex commutative diagrams for which the earlier available LaTeX tools (XYpic) were weak had to be drawn in a vector graphics editor. Drawing of one chart took 2-3 hours. After the appearance of more advanced tools (TikZ) in LaTeX [179, 180], the production of the same diagram requires no more than 15 minutes.

There are also scripting languages for computer algebra software like commercial Maple [181] and Mathematica [182], or free Maxima [183, 184]. These languages describe sequences of operations.

For mathematics, simple formulas can be OCRed, but OCR systems fail on complicated formulas. The problem was solved after introduction of Deep Learning. Online platform MathPix [185], that has inexpensive license and a number of free access permissions, is today a standard generally accepted solution. Mathpix recognizes text in 32 languages, mathematics, and chemical

diagrams. It also recognizes handwritten mathematics, chenical diagrams, all Latin alphabet languages and Hindi.

Besides this, a number of Deep Learning open source solution can also provide accurate result if there is computing resources and proper dataset to train them[186].

7.6 Chemistry

One important challenge in chemistry is the unequivocal identification of chemicals for production and trade, given the vast assortment (millions) with multiple trade names and numerous image variants, especially for structurally complex chemicals.

International industry standards in chemistry are established by IUPAC (International Union of Pure and Applied Chemistry [187]). These standards include IUPAC nomenclature for inorganic and organic chemistry. Additionally, InChI (IUPAC International Chemical Identifier) is supervised by InChI Trust [188]. InChI represents a molecule as a unique text string, and each molecule's description is unique. An example is the InChI for ethyl alcohol (CH3CH2OH):

InChI = 1S/C2H6O/c1-2-3/h3H,2H2,1H3

A hash code (InChI key) of constant length is generated from the InChI string, and InChI keys, like LFQSCWFLJHTTHZ-UHFFFAOYSA-N for ethanol, are used for chemical database searches.

InChI has expanded beyond molecules to describe chemical reactions through the RInChI project [189].

Like InChI, IUPAC nomenclatures provide unambiguous descriptions of molecules but in a more human-readable form.

Other scripting languages for writing chemical formulas include LaTeX, which offers over 55 packages for chemistry [190], and SMILES (Simplified Molecular Input Line Entry System [191]), allowing both molecular formulas and reaction equations.

MDL MOLfiles [192] provide a specific scripting notation with coordinates of atoms in the 3D structure of the molecule, but it lacks unambiguity.

For recognizing chemical formulas and equations in linear form, tools like Mathpix can be employed.

Recognizing images of chemical (molecular) structures is a long-standing text recognition task. Solutions like OSRA[193] and IMAGO[194] have gained popularity due to their advanced features, including support for various operating systems and online/desktop versions. A recent application, MolVect[195], has gained popularity for its jar-based solution with advanced features.

Deep learning techniques have also introduced widely used solutions, with DECIMER[196] emerging as a competitor to previously popular solutions.

In recent years, many free and commercial solutions for chemistry using optical character recognition have been developed. Another application is reviving the chemical structure recognition task, stemming from the modern establishment of stereochemistry standards [197] for any organic chemistry publication.

To address the recognition of chemical molecular structure images in vector formats, the IMAGO [194] and MolVec [195] tools, both recognizing in the Mol format, were chosen for the server side of the HeDy platform. These tools were selected due to their implementation as jars, making them cross-platform and easy to use.

The choice for the frontend proved more complex due to the variety of free online chemical structure editors. Consulting with scientific researchers in organic chemistry led to the selection of the Ketcher editor [198] for the HeDy interface. This editor meets specific requirements for editing recognized chemical structure vectors, including a stereo element toolbar and a 3D editing mode.

7.7 Technical drawing

We should mention here languages used with CAD software.

In mechanical engineering and building, the most known CAD is AutoCAD. This is a commercial system, and its language doesn't even have open specifications as it is used interactively. However, there are a lot of

CAD products including free open-source ones, which makes obvious the necessity to develop standards to exchange CAD projects in the electronic form. For example, it was estimated [199] that as on 2004 the USA capital facilities industry had lost annually $15.8 billions due to inadequate interoperability arising from "the highly fragmented nature of the industry, the industrys continued paper based business practices, a lack of standardization, and inconsistent technology adoption among stakeholders".

Several project exchange languages for CAD are standardized on international or industry level. All are scripting, although some of them have two variants: textual (scripting) and binary. Thanks to this, CAD not only provides a graphical representation of design objects, but also, for example, strength calculations use the same script as the source code. It is also possible to generate commands for CNC machines or 3D printers. This proves the necessity and usefulness of the standard script description as opposed to any other.

Examples of manufacturer-neutral scripting standards for CAD file exchange are IGES and STEP (ISO 10303, Standard for the Exchange of Product model data). IGES scripts consist of 80-character records to be punched on cards. Since the initiation of STEP in 1994, IGES stopped its development, its last version being dated by 1996. Nevertheless, old IGES files since 1980s can be reused today.

Let us take the concept of BIM (Building Information Modeling, or Building Information Model). In this concept, the only information model of the building is provided by a single script. It is used at all stages, from design and construction to the operation of the building inclusive. A three-dimensional model of a building object is associated with an information database, in which additional attributes can be assigned to each element of the model. The building object is actually designed as a whole. Changing anyone of parameters entails an automatic change of the other parameters and objects associated with it, up to drawings, visualizations, specifications, financial calculations and calendar schedules.

For graphical design content, the situation with recognition is similar to that for diagrams and drawings in chemistry and mathematics.

The recognition of technical drawing documents is part of CAD industry.

CAD professionals use commercial systems, which are expensive but provide 99% accuracy for example[200].

The majority of these systems allows disposable free access, for example [201] or trial version for example [202], [203]. So, we can omit processing of examples of CAD drawing, when such elements exists in tutorials or scientific papers, and address to one of these systems.

Heterogeneous content types of technical drawing, which are usually met in printed documents, are graphs and flowcharts. For these types of heterogeneous content, the main purpose of digitization is the reuse. The task of reuse arises mostly once, so modern solutions are either open source or online systems. Another specific feature is that output format has to be suitable for reuse. Modern systems chose formats of vector editors or of diagram creation software. The employment of Deep Learning supplies also the output of shape containers, for example: Python dictionaries or JSON, which can be easy convert to any required format. We tested several open source GitHub solutions. For our task the most accurate results were obtained by Handwritten-Flowchart-with-CNN [204] and Image2CAD[205] projects. These solutions are based on Deep Learning and provide output in the form of Python dictionaries.

7.8 Charts

Charts are important and omnipresent elements of any kind of heterogeneous documents: personal, business and scientific. The digitization of charts mainly consists in chart type classification and extracting of dataset. Having type and dataset, the chart can be easily reconstructed due to the existing numerous software. Charts wide spreading supposes that the digitization tools exist for any specific problem. However, most tasks suppose digitizing charts whose type is known. In consequence of this, the classification task was separated as an optional subtask like popular ChartReader[206] or even the main task of the project like Chart-Image-Classificatio[207]. Digitization of particular type of charts is today ordinary task. The solutions can be obtained by a number of tools both online and desktop. Some popular desktop solutions like PlotDigizer[208] and EngaugeDigitizer[209] are not

only free and open source, but included in many package managers like Python PIP. Despite this fact, the WebPlotDigitizer[210] online solution is often used today because of its convenience and accuracy. Many developers propose the interfaces to work with WebPlotDigitizer at desktop. In addition to the available software, personal solution can be developed using of free libraries of several programming languages, mostly Python. The popular commercial software systems also have today chart digitization tools. The most known example is plot digitization option in MS Excel.

7.9 Chess

The standard script language for chess is PGN (Portable Game Notation [211]). PGN may describe positions, moves, and possible move variations. PGN scripts contain metadata like name of the tournament, names of players, date, etc. Example of PGN is shown in Tab. 7.1.

Table 7.1: Portable Game Notation for chess

```
[Event "F/S Return Match"]
[Site "Belgrade, Serbia JUG"]
[Date "1992.11.04"]
[Round "29"]
[White "Fischer, Robert J."]
[Black "Spassky, Boris V."]
[Result "1/2-1/2"]
```

Position after 14. Bg5...

```
1. e4 e5 2. Nf3 Nc6 3. Bb5 a6 4. Ba4 Nf6 5. O-O Be7
6. Re1 b5 7. Bb3 d6 8. c3 O-O 9. h3 Nb8 10. d4 Nbd7
11. c4 c6 12. cxb5 axb5 13. Nc3 Bb7 14. Bg5 b4 15. Nb1 h6
16. Bh4 c5 17. dxe5 Nxe4 18. Bxe7 Qxe7 19. exd6 Qf6 20. Nbd2 Nxd6
21. Nc4 Nxc4 22. Bxc4 Nb6 23. Ne5 Rae8 24. Bxf7+ Rxf7 25. Nxf7 Rxe1+
26. Qxe1 Kxf7 27. Qe3 Qg5 28. Qxg5 hxg5 29. b3 Ke6 30. a3 Kd6
31. axb4 cxb4 32. Ra5 Nd5 33. f3 Bc8 34. Kf2 Bf5 35. Ra7 g6
36. Ra6+ Kc5 37. Ke1 Nf4 38. g3 Nxh3 39. Kd2 Kb5 40. Rd6 Kc5
41. Ra6 Nf2 42. g4 Bd3 43. Re6
1/2-1/2
```

Notations for some specific tasks are FEN (Forsyth-Edwards Notation)

and EPD (Extended Position Description), which are defined in [211, Sec. 16].

LaTeX permits to generate chess diagrams and game notation. Search [212] results in 21 packages for this purpose.

There are several tools to recognize chess diagrams for the purpose of further position analysis. The first one is ChessOcr [213] for Android dated back to 2015. ChessOcr exports its results in a PGN file, and puts the most recent position in the clipboard as a FEN string. This is a simple scripting format to record chess games.

Chessify [214] is a modern commercial AI-based application for Android and MacOS with rich set of features. It is positioned as the ultimate supporting tool for any chess player. Between their features we find optical recognition of chess books in PDF, and recognituion of chess diagrams including photos of real chessboard.

7.10 Final remarks

More scripting languages exist, e.g., for geographic maps: Geography Markup Language (GML) – XML based; older VRML [215] and newer X3D [216] for 3D objects, etc.

And, finally, it is not clear what to do with the results of such recognition. How to integrate them into the script image of the page?

Integration of heterogeneous content recognition results can be solved by export or conversion of those in the same format. The first candidate for this role is LaTeX but the implementation may require significant improvements in it. The second obvious option is to use XML-based formats.

Chapter 8

Platform for recognition of heterogeneous documents

8.1 Introduction

In this chapter, we present a web platform, or a digitization platform, that evolved along two branches. The first branch is tailored for documents with homogeneous content, specifically old Romanian Cyrillic texts. The second branch expands the functionality of the first, enabling the processing of documents with heterogeneous content [8, 160]. Modules for page structure analysis, page segmentation into homogeneous fragments, and integration of tools for recognizing not only texts but also music, formulas, and more, have been added to this branch.

The first branch includes modules to perform four main tasks related to digitizing old Romanian Cyrillic and other language documents with homogeneous content. This encompasses image preprocessing, document recognition, transliteration of recognized text from Cyrillic to Latin script, and the management and publication of digitized documents. The platform can be used both as a web application and as a desktop application.

The second branch extends the functionality of the first one introducing steps for page layout analysis, recognition of non-textual page fragments, and assembly of the resulting document. Both branches share common modules

for image preprocessing, text recognition and more. The second branch provides additional tools for processing documents with heterogeneous content.

In cases of heterogeneous content, the platform incorporates functionality for assembling the resulting document from diverse scripts obtained through the recognition of heterogeneous content. This process is realized by incorporating script files as applications into a PDF file. Thus, the platform integrates various scripts into a single document, offering users a convenient tool for compiling and formatting digital materials.

The generalized functionality of a platform for recognition of heterogeneous documents is presented in Tab. 8.1.

Table 8.1: Recognition of heterogeneous documents: platform functionality

Input data	**Process**	**Resulting data**
A: Graphical document (paper, photocopy, etc.)	**(P1)**: Imaging/scanning	**B**: Page image(s) in electronic form
B: Page image(s) in electronic form	**(P2)**: Image quality improvement	**B**: Page image(s) in electronic form with better quality
B: Page image(s)	**(P3)**: Page layout (structure) analysis	**C**: Page map(s) **D**: Page fragment(s) (smaller files)
D: Page fragment(s)	**P4**: Fragments recognition according to type(s) of fragment(s)	**E**: Script equivalent(s) of page fragment(s) **F**: Extracted metadata

Continued on next page

Table 8.1: Recognition of heterogeneous documents: platform functionality (Continued)

Input data	Process	Resulting data
E: Script equivalent(s) of page fragment(s) **F**: Extracted metadata **C**: Page map(s) **D**: Page fragment(s) **B**: Page image(s)	**P5**: Task distribution for manual verification	
E: Script equivalent(s) of page fragment(s) **F**: Extracted metadata **C**: Page map(s) **D**: Page fragment(s) **B**: Page image(s)	**[P6]**: Human verification; correction if necessary	**E**: Script equivalent(s) of page fragment(s) **F**: Extracted metadata
E: Script equivalent(s) of page fragment(s) **F**: Extracted metadata **C**: Page map(s) **D**: Page fragment(s)	**P7**: Assembly of script presentation of page(s) and metadata integration	**G**: Script equivalent(s) of page(s)
G: Script equivalent(s) of page(s)	**P8**: Reconstruction of page image(s) from the script	**H**: Reconstructed page(s)
H: Reconstructed page(s) **B**: Page image(s)	**P9**: Automated verification	**I**: Verification log(s) **J**: Error report(s)
H: Reconstructed page(s) **B**: Page image(s) **I**: Verification log(s) **J**: Error report(s)	**P10**: Task distribution for manual verification	

Continued on next page

Table 8.1: Recognition of heterogeneous documents: platform functionality (Continued)

Input data	Process	Resulting data
G: Script equivalent(s) of page(s) **H**: Reconstructed page(s) **B**: Page image(s) **I**: Verification log(s) **J**: Error report(s)	**[P11]**: Human verification; correction if necessary	**G**: Script equivalent(s) of page(s) for further use

A-J: letters permit to follow data usage
P4: automated processes
(P1): semi-automated processes with little or medium manual intervention
[P6]: manual processes

8.2 General structure of the HeDy platform

The HeDy platform is developed based on advanced computational methods and technologies derived from our experience in natural language processing, machine learning, and web development.

As we noted above (Sec. 8.1), the platform comprises two branches: for documents with homogeneous and heterogeneous content. Despite their distinct focuses, these branches share some functional modules.

The HeDy platform is presented as an application with an interactive React-based GUI and a set of APIs managed through Django and Django Rest Framework, connecting the GUI to tools and resources for digitizing Romanian Cyrillic documents. Users can create their digitization applications using existing tools or by adding new ones, such as OCR models, image processing modules, or word dictionaries. The graphical interface extends the Stepper component, providing a wizard-like workflow.

Access to a variety of digitization tools is a key feature, facilitating the digitization [160].

The platform toolkit includes image pre-processing/processing mech-

anisms like Scan Tailor, ABBYY FineReader, and OpenCV. Additionally, it incorporates an image slicing module into fragments with homogeneous context, and virtual keyboards for alphabets such as Romanian Cyrillic, Transitional Alphabet, and Moldavian (Soviet) Cyrillic.

The integration of these tools into a single platform aims to provide users with a unique graphical interface, enabling control over the entire document digitization process.

The software part of the platform is available on Github [217]. The web application allows the development and integration of separate Python modules. These modules can run independently, with Django serving as a hub connecting the UI to the outputs of the required modules.

The digitization platform addresses several key tasks, including:

Image Pre-processing. Currently offering two main preprocessing modules (image binarization and image resolution correction), with additional modules integrated through third-party applications like ScanTailor and FineReader.

Splitting Heterogeneous Content. Dividing heterogeneous content into homogeneous fragments and categorizing them based on content types.

Optical Content Recognition and Editing. Employing an OCR system for Romanian historical documents, based on pre-trained models spanning the 17th-20th centuries. Administrators can add new models, with OCR templates provided in FineReader XML format.

Post-Recognition Processing. Involving tasks such as transliteration of recognized text, editing the text obtained during transliteration, and saving the results.

The digitization platform is structured into functional groups that facilitate specific tasks:

Image Processing Group. Includes basic preprocessing modules (image binarization and resolution correction) and additional modules integrated with third-party applications.

Document Recognition Group. Involves selecting OCR models, using word dictionaries for recognition, editing recognized text, and incorporating OCR exception dictionaries.

Text Transliteration Group. Manages the transliteration of recognized

text, updating spelling, using exception dictionaries, and editing text post-transliteration.

Management and Publication Group. Covers saving recognized/transliterated texts, downloading processed images, uploading original documents to the cloud, saving digitized object status, and expanding exception dictionaries. It also includes a module for the publication of digitized documents, considering factors like copyright laws, document conservation, and public access.

Fig. 8.1 illustrates the structure of the digitization platform, showcasing the four functional groups (G1–G4).

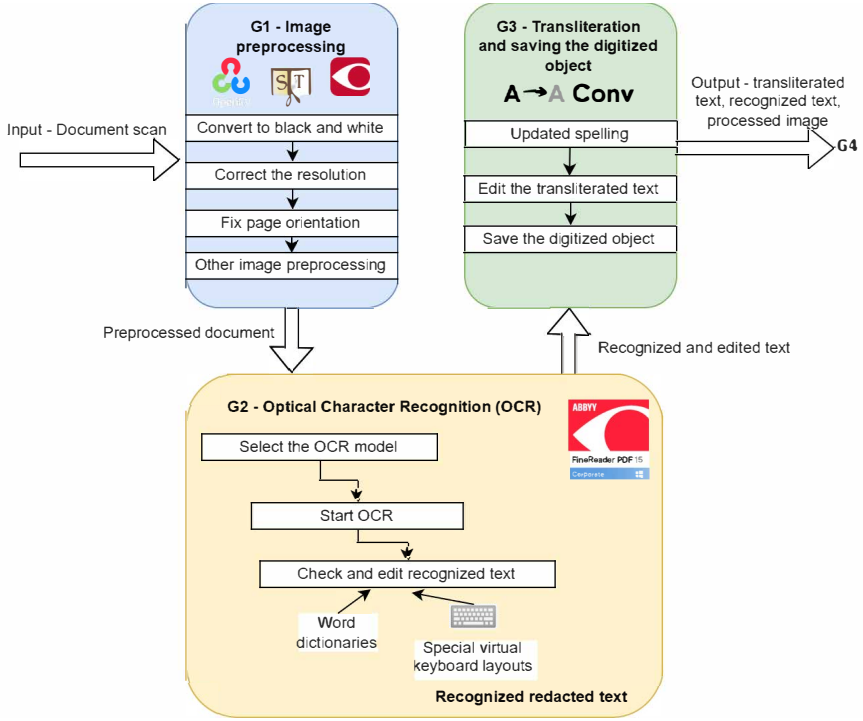


Figure 8.1: The structure of the platform.

8.3 Platform architecture details

The digitization platform contains a set of modules that facilitate the tasks of image processing, dividing an image with heterogeneous context into homogeneous parts, recognizing homogeneous parts of documents, assembling them into a single document, saving the result. These modules are organized into functional blocks or groups. There are also common modules that are not explicitly included in functional groups.

The first functional group deals with image processing. This group is equipped with two basic preprocessing modules: image binarization and resolution correction, which are extended by additional modules due to integration with third-party applications such as Scan Tailor, FineReader, OpenCV, GIMP, etc.

The second functional group contains modules for optical document recognition. This group includes: splitting an image with heterogeneous context into homogeneous parts, recognizing homogeneous parts of documents, editing recognized text.

Optical recognition mechanism is based on FineReader 15 and other special programs for recognition of mathematical texts, notes, etc. There is a set of trained models. For example, for text the models are presented in FineReader XML format (.fbt files) and contain OCR model configurations, a set of training data, the necessary alphabet, and word dictionaries.

The third functional group contains components related to the layout of the whole document, its checking by the user.

The fourth functional group contains modules for managing and publishing digitized documents. Includes: saving recognized/transliterated texts in various formats; uploading processed images; uploading original documents and processed images to the cloud ; saving the state of the digitized object in the database; expanding exception dictionaries based on retrieved texts.

The graphical interface of this application extends a component Stepper [218] which displays the progress through numbered steps, providing a workflow like a wizard [219]. The digitization application allows for the recognition of Romanian Cyrillic documents from the 17th to 20th centuries, the transliteration of texts into Latin script, the editing of recognized/transliterated texts, as well as the downloading or publishing of the results.

8.4 Image preprocessing modules

The initial functional group, denoted as G1, is dedicated to image preprocessing, and its modules are shared between both branches of the platform. The aforementioned challenges related to this aspect were previously explored in detail in Chapter 2. For the sake of clarity and convenience, we will provide a brief recapitulation of the primary issues here

In the context of optical recognition over text and other types of content, preprocessing typically refers to the steps taken to prepare a text image for analysis by the OCR engine. The OCR engine may sometimes struggle to correctly interpret images that are blurry, distorted, or have low contrast. Preprocessing can help improve OCR accuracy by preparing the image to be more suitable for recognition. Some common preprocessing steps for OCR include:

Image Quality Enhancement: This may involve techniques such as adjusting the image's contrast or brightness to improve readability, or sharpening the image to reduce blurriness.

Binarization: This involves converting the image to black and white, which can help improve contrast and make text recognition easier.

Noise Removal: This can involve removing extra black pixels from the image that might create obstacles in the functioning of the OCR engine.

Distortion Correction: If the image is not perfectly aligned, the OCR software may have difficulty correctly interpreting the text. Correcting distortion involves rotating the image to align it properly.

By preprocessing the image before sending it to the OCR engine, the accuracy and reliability of the OCR process can usually be improved.

In this functional group, preprocessing modules from software such as Scan Tailor, FineReader 15, and the Python package OpenCV are integrated. These tools are presented in Tab. 8.2.

8.5 Optical Character Recognition (OCR) modules

The modules in this functional group are used for managing OCR models, actual document recognition, and editing the recognized text. Recognition

Table 8.2: Image preprocessing modules and integrated software for implementing the modules in the functional group G1.

Image preprocessing module	**Integrated engines/tools to implement the module**
Selection of the preprocessing engine	Scan Tailor, FineReader 15, OpenCV
Image binarization	Scan Tailor, FineReader 15, OpenCV
Manual image resolution setting	Scan Tailor, OpenCV
Automatic page orientation correction	FineReader 15, OpenCV
ISO noise reduction	FineReader 15, OpenCV
Image splitting into multiple pages	FineReader 15
Automatic image resolution correction	FineReader 15
Straightening text lines	FineReader 15
Manual page orientation correction	Scan Tailor
Removing image stains	Scan Tailor
Image illumination correction	Scan Tailor
Managing character thickness	Scan Tailor

of non-textual fragments for heterogeneous pages will be described later in this chapter.

The actions in group G2 begin with selecting the document's historical period. Typically, the user knows the historical period of the document to be digitized. Moreover, the user can even specify the year of publication. This information can be obtained from the book cover, the source from which the document was obtained, by checking the document's history if it has a known history, or by analyzing the style and type of writing. For example, if the document contains both Cyrillic and Latin letters in the same word, it may date from the period between 1830 and 1860 when transitional alphabets were used.

Additionally, examining any dates mentioned in the document or consulting an expert in the field can provide insights into its historical period. However, there are cases where the user has a few random images from a document and doesn't know anything more about it than the fact that it's in Cyrillic script. For such cases, an automatic period detection module would

be useful and is planned to be developed within the platform. One approach to solving this problem is based on the experience of font detection in Cyrillic documents printed in the 17th century, where certain neural network models were trained to automatically recognize the document's font. Such a module that handles the detection of fonts from the 17th century is included in G2.

The most important module in G2 is the selection of the OCR model. The user has the option to choose an OCR model from those included by default or to add a new model according to the platform's established conditions. By default, a total of 8 OCR models are included. There is one model for the 20th century, 2 models for the 19th century, 3 models for the 18th century, and 2 models for the 17th century. These OCR models were obtained by training the OCR engines from FineReader 12 and FineReader 15.

The model for the 20th century extends through the transfer learning The OCR model for the Russian language integrated by default in the FineReader 15 language package was primarily trained to recognize the letter ѫ, which does not exist in the Russian alphabet. The training dataset consisted of 1040 training examples and was based on the following sources: the newspaper *Literatura și Arta* from the years 1988-1989, the magazine *Femeia Moldovei* from the years 1960-1970, and the book *Folclor din părțile Codrilor* from 1973. An important advantage of this model is its dictionary with over 447,000 words. This dictionary includes both word roots and their inflected forms. The generation of the word dictionary for this model is based on a backtracking algorithm that generates the lexicon in Cyrillic script, using certain rules of reverse transliteration, where modern Romanian words written in Latin script were transposed into their equivalents in Cyrillic script. The character-level accuracy of this model exceeds 98%.

Considering that documents in the 19th century were printed in different alphabets, two OCR models were trained, one based on the Romanian Cyrillic alphabet and the other based on the transitional alphabet. The first model, trained with characters from the Romanian Cyrillic alphabet, is based on datasets from the books *Legiuire* from 1818 and *Epistolariu românesc* from 1841. The training dataset contains 2800 examples, and the word dictionary has over 3000 words. The second model was trained on datasets from documents printed with the transitional Cyrillic-Latin alphabet. In particular,

Gheorghe Asachi's book *Elemente de aritmetică* from 1836 was used for this purpose.

The models for the 18th century were trained using datasets taken from the following documents: *Fiziognomie* from 1785; *Așezământ* from 1786; and *De Obște Geografie* from 1795. The word dictionary is common to all three models and contains over 3000 words. In particular, the model based on the book *Fiziognomie* was trained with 3600 training examples and has a dictionary of 1800 words.

The training and evaluation of OCR models for the Romanian Cyrillic alphabet used in 17th-century prints were described in Sec. 4.5. Specifically for this case, the font problem was identified, and classification solutions were proposed. We will mention here that the baseline model for the 17th century was trained on a dataset consisting of 3668 training examples extracted from the New Testament from Bălgrad from 1648, accompanied by a dictionary of 4582 words. The accuracy (at the character level) of this model is approximately 95%.

The process of document recognition can be divided into several parts that can be performed in parallel to improve efficiency (processing speed for multiple documents). The use of ABBYY Hot Folder (continuing with Hot Folder), an agent that allows the application of the necessary OCR model to a folder of images, which will be automatically processed by the FineReader 15 OCR engine when new images appear in the folder, can help parallelize the process. This can be achieved by using multiple instances for each OCR model, thereby dividing the preprocessed images into multiple folders, which can improve efficiency when multiple users are working simultaneously on the platform. The execution of the loaded models on the platform is managed through the Hot Folder agent.

Recognizing a single page of text on average takes 30 seconds, although sometimes recognizing such a page can take up to two minutes. This is because the instances created in the Hot Folder check every minute (this is the minimum time option in Hot Folder) if new processed images have appeared in the processed image folders. Therefore, if a document was processed in the 55th second, the OCR process will start in 5 seconds, and if a document was processed in the 5th second, it will have to wait for 55

seconds before the OCR process starts. A PDF document with 50 pages of text will be recognized in about 90 seconds; a PDF with 100 pages of text took 150 seconds; a PDF with 360 pages of text took over 385 seconds (more than 6 minutes). Text documents in PDF format take approximately 1.2 seconds per page. For PDFs with images, there is no consistent duration observed. The criterion for OCR accuracy at the character and word levels is analyzed in Chapter 5. For example, the OCR model for the 20th century provides an accuracy of over 98% at the character level; models from the 18th century offer over 92% accuracy at the word level; and the model for the 17th century provides an accuracy of over 95% at the character level and word dictionaries, considering proper image preprocessing, document scanning quality, wear and tear, etc.

In addition to the word dictionaries used within the OCR engine, the platform also includes OCR exception dictionaries consisting of tuples formed by an expression containing recognition ambiguities and the correct variant of that expression. The term *recognition ambiguities* is used simply because some letters have extremely close graphic similarities, and sometimes the OCR engine recognizes the wrong variant with very high probability. In such cases, the internal dictionary cannot propose the correct candidate even if the correct variant were in the dictionary. For example, the letter **и** is confused with the letter **н** in the expression **сърачїн**, and the OCR exception dictionary could contain the exception: (сърачїн, сърачїи). To address such situations, we have included a post-processing component in G2 that uses the exception dictionary. Exception dictionaries are also used for transliteration, and a similar module is available in G3.

Within the functional group G2, we have included a module for editing the recognized text. This text editor features a web-based virtual keyboard that adapts its character composition based on the document's historical period, relying on the JavaScript simple-keyboard. There is also a module dedicated to managing virtual keyboards for the desktop. By default, we have a virtual keyboard loaded for Windows with Romanian Cyrillic characters.

The spell checker in the text editor is based on the JavaScript simple-spellcheker and the dictionaries used in OCR modules. In addition to the integrated spell checker in functional group G2, some browsers like Mozilla

Firefox and Google Chrome offer their own spell-checking services, but they are not yet useful for texts in Romanian language written in Cyrillic script because they do not allow the addition of custom dictionaries. At least these services are useful for transliterated text.

8.6 Text Transliteration Modules

The G3 group includes modules for transliteration and editing of transliterated text. Transliteration can be done through two methods. The first method is using the web-based transliteration application AAConv [220], and the second option is using the same application but in a desktop version. A notable difference between these two variants is that the web version can only accept limited volume of text in a single processing.

The historical period of the document retains the state from G2 when the user uses the selection module; otherwise, the period selection is accessible from G3. The document's historical period is, in fact, an attribute of all modules in both G2 and G3.

An important module for the user is the spelling update module, where, upon request, the modern Romanian language spelling norms are considered. An example is the use of **â** (from **a**). In the transliteration process, the transition to writing with **â** is achieved through transliteration, only if the spelling update option is activated; otherwise, the original spelling is retained. It should be noted that according to the recommendations of the Romanian Academy, the letter **î** will always be written at the beginning and end of a word (**început**, **înger**, **în**, **întoarce**, **a coborî**, **a urî**). Inside a word, **â** is usually written (**cuvânt**, **a mârâi**). However, there are some exceptions to this rule. Prefixed words that start with the letter **î** will retain the **î** inside. For example, **neîmpăcat**, **neîngrijit**, **preîntâmpinat**, **reînarmat**. The same rule applies to compound words: **bineînțeles**, **semiînchis**, etc. [221].

During transliteration, we may encounter deviations from the general rules that cannot be controlled through preset rules, and for this reason, group G3 includes a module for using the exception dictionary. The transliteration exception dictionary contains words that cannot be correctly transliterated using only transliteration rules. For example, the word **амязэ** according to

transliteration rules is transformed into **amează**, while the correct variant is **amiază**, which is found in this dictionary. In this module, it is possible to manage the list of exceptions. Exceptions are handled after transliterating the text from Cyrillic script to Latin script according to the rules but before viewing and verifying the text in the text editor. Many exceptions come from the different spelling of foreign-origin words, especially proper nouns.

Additionally, group G3 shares the same text editing module with group G2, and the virtual keyboard and word dictionaries for the spell checker are adapted to transliterated text. It should be noted that the virtual keyboard contains letters from the modern Romanian alphabet, and the word dictionary is written in the modern Romanian alphabet.

An experimental module is the correction of transliterated text using a state-of-the-art artificial intelligence system. This system is called GPT-3 developed by OpenAI [222]. The machine learning models, also known as linguistic models, in GPT-3 can solve natural language processing tasks such as generating summaries, paraphrasing text, automatic translation, text classification, converting natural language text into programming language code, text correction, and more. GPT-3 performs these tasks with very good accuracy, both in English and Romanian. In the module implemented in G3, we use the *text-davinci-003 model* [223] (referred to as Davinci) for text correction by providing the condition: correct the text X, where X is the transliterated text. The model can process up to 4000 tokens per request. One token in Davinci, on average, consists of about 4 English characters, and 100 tokens would roughly equate to approximately 75 words. An important point to note about correction is that this model does not preserve the original version of archaic expressions from the transliterated text. For example, Davinci corrected the expression *"Knowing and understanding Your Majesty, we Romanians who are in Your Majesty's land, do not have either the New Testament or the Old Testament in our language."* to the transliterated expression *"How can one know that, seeing and understanding Your Majesty, we Romanians who are in Your Majesty's land. We do not have either the New Testament or the Old Testament fully in our language."* Given that this text was written in the 17th century, what GPT-3 has done is more akin to aligning the old text with modern text. However, this is not the case for texts from

the 20th century, where the text differs insignificantly from modern texts. In this case, Davinci corrects quite well. As mentioned earlier, text correction with GPT-3 is an experimental module that needs more exploration to draw more general conclusions.

Other modules in G3 deal with stylistic correction of the transliterated text. This may include replacing the apostrophe with a hyphen (example: **s'ar** with **s-ar**) or removing the hyphen from a word at the end of a line (example: **ră- pirea** becomes **răpirea**).

8.7 Modules for managing digitized documents

The digitized document refers to the original and preprocessed images, the recognized texts, and the transliterated ones. The functional group G4 includes modules for managing the digitized document. Modules described below are used in both branches of the platform.

A module in G4 is the downloading of preprocessed images. The user can download the images to their personal device for later needs. The format of the downloaded images is JPG.

Similar are also the modules for downloading recognized and transliterated texts. The formats of the downloaded files include TXT and DOCX. The DOCX version is nothing but a packaging of the raw text (without formatting), without retaining the styles or illustrations from the original document.

An important module in G4 is saving the digitized document in the platform's database. In addition to storing texts and links to files, a digitized object is also stored, which represents a JavaScript object that helps to preserve the state of each step taken through the digitization application described in the following section. The object includes preprocessing parameters, recognition and transliteration parameters, the recognized and edited text, the transliterated and edited text, etc.

To cope with the multitude of images uploaded to the platform, we have included in G4 a module for uploading original documents and preprocessed images to the Cloud. In developing this module, we created a bucket using the Amazon S3 service [224] (Simple Storage Service), a cloud storage service developed by Amazon Web Services (AWS). An Amazon S3 bucket is a con-

tainer for storing files in Amazon S3. We can store any type of file in a bucket, from images, music, and videos to applications, websites, etc. A bucket is identified by a unique name in Amazon S3 and can be accessed via a URL that starts with "https://s3.amazonaws.com/". It can be configured with various security options to protect its content and can be used to distribute content through Amazon CloudFront [225], a content delivery network (CDN).

A set of very important modules in G4 refers to the publication of the digitized document. Being aware of the issue of copyright and opting to solve it while respecting legal regulations, in the following, we will propose only technical solutions that could work assuming the resolution of legal aspects. The publication module is focused on the eMoldova portal [226, 227], especially based on a portlet named the National Digital Treasury [228] (continue Digi). This portlet contains resources from the Moldovan digital treasury, as well as some simplified services for digitizing and managing digitized documents (called digital articles in Digi). Digi includes special working groups to provide the public with high-quality digital articles in terms of the accuracy of the recognized/transliterated text. The working groups are divided into two. There is the digitizer working group, those who upload a digitized document to the platform, and the editor working group, those who check and approve the digital article for final publication. For metadata labeling, Digi offers a large selection of tags, which come from the metadata of different publications. This allows for simple searching and filtering of articles based on the type of document (magazines, books, manuscripts), a specific year or period. Internal tags are also included in labeling. Articles marked with these tags have limited access, being visible only to users with special roles or privileges. An example is the *For editing* tag, which is used for articles uploaded by a digitizer. Articles tagged with this tag are only accessible to the group of editors. When an editor approves a digital article for publication to all users, the *For editing* tag will be replaced with *Verified by editor*. Digi plans to add a condition that several editors must approve a digital document for final publication.

Other features of Digi include: notification of work groups; creation of article drafts; versioning of edits made to a digital article; adding bookmarks for efficient monitoring of user-focused articles; a system for rewarding users

for their activity; appreciation of digital articles; adding comments on a digital article or another; merging several digital articles into a single digital article (which can be useful when digitizing a book or magazine), etc. In Fig. 8.2, a page [229] of such digital article is presented, which includes the original document, the recognized text, the transliterated text, and a side menu on the right side of the page to switch the desired content.

Figure 8.2: Page of a digital article published in Digi

8.8 Digitization process overview for historical documents

The digitization application is a practical showcase of our platform's capabilities. It exemplifies various modules and their functionalities through a 7-step digitization cycle for historical Romanian documents. The graphical interface in Fig. 8.3 on p. 138 illustrates three states of the digitization cycle.

Step 1. Uploading Files:

- Acceptable file types: png, jpeg, tiff.
- Total size limit: 700MB; individual file limit: 100MB.
- Multiple files processed in a cycle should share the same characteristics.

Step 2. Preprocessing of Uploaded Images:
- Choose from Scan Tailor, FineReader 15, or OpenCV engines. - Scan Tailor (desktop) offers extensive options; Scan Tailor (web) streamlines basic preprocessing.
- FineReader 15 handles resolution correction, orientation, black-and-white conversion, noise reduction.
- OpenCV provides manual resolution setting and integrated filters for cleaning images.

Step 3. Optical Character Recognition (OCR):
- Select document period and recognition model (Fig. 8.4 on p. 138).
- Example output displayed in Fig. 8.5 on p. 139.

Step 4. Verification and Editing of Recognized Text:
- Manual text processing with virtual keyboard and word dictionaries (Fig. 8.6 on p. 139).

Step 5. Transliteration of Recognized and Edited Text:
- Integrates transliteration modules and exception dictionaries (Fig. 8.7 on p. 140).

Step 6. Verification and Editing of Transliterated Text:
- Similar to the previous step, operates with the same dictionaries.

Step 7. Saving the Results:
- Download recognized text, transliterated text, and preprocessed images using modules from the G4 group (Fig. 8.8 on p. 140).

Please note that the user can skip certain verification and correction steps based on their needs, ensuring flexibility in the digitization process.

8.9 Digitization steps for heterogeneous content

The Heterogeneous Content Platform integrates many tools for such processing and allows converting images of printed pages into a set of texts and scripts depending on the content type. Unrecognizable parts of the page remain in the image format. The resulting page fragments can be reconstructed from the original image. Fig. 8.1 on p. 125 shows the structure of the platform.

The following types of fragments are provided: images, text, notes, math-

Figure 8.3: Three states of the document digitization cycle

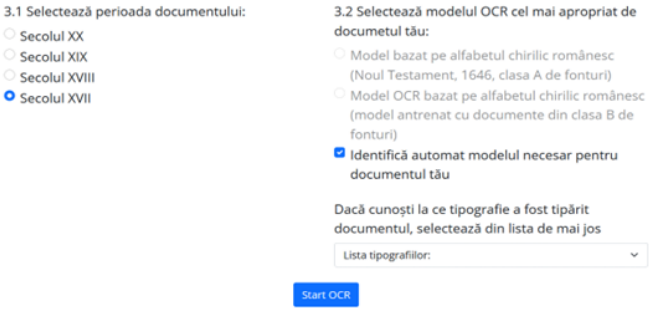


Figure 8.4: Available options in step 3

ematical formulas, chemical formulas and structures, chess diagrams.

The platform implements a user interface that serves as a bridge between the user and a variety of tools for solving specific subtasks (see below), including third-party developments.

The platform is extensible and uses convergent technology to seamlessly integrate external subsystems [7]. The platform supports step-by-step processing of heterogeneous documents. Each step performs a completed operation, the results of which can be passed to the next step or used directly. Some steps can be skipped or repeated.

Document processing starts by loading one or more page images. PNG, JPEG, GIF, TIFF and PDF files are supported. You use a standard dialog box or drag and drop to select files. The loaded pages are displayed on the screen.

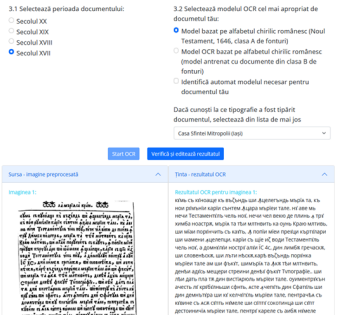


Figure 8.5: Execution of step 3 in the digitization application

Figure 8.6: Verification of Recognized Text

The next step is to refine the images so that they are ready for recognition. Tools for such processing are external programs such as Open CV, FineReader, ScanTailor, Gimp, ImageMagick, selectable through the menu.

If there are pages with heterogeneous content, image fragmentation is performed. It segments page images basing on content type with manual intervention. Our Python program for image fragmentation uses the FineReader Engine (FRE) command line interface to analyze page layout. The resulting XML file contains coordinates, content types (text, image, table, separator, etc.), and recognized text for text segments. The restructuring module enables the reconstruction of fragment geometry.

To address limitations in FRE v.12, we have introduced manual intervention capabilities. Users have the option to manually segment the image,

Figure 8.7: Transliterating Recognized Text

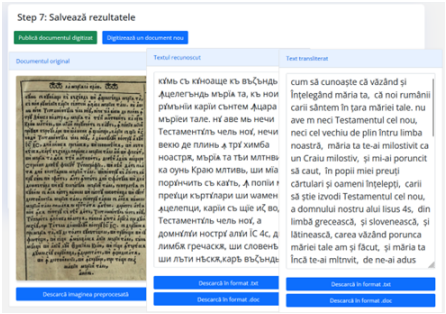


Figure 8.8: Step 7 in the digitization application

correct segment boundaries, and set segment types. This manual intervention provides a method to refine the segmentation process.

At this stage, image areas with homogeneous content are selected and their type is determined. For this purpose, ABBYY FineReader Engine is called via a Python middleware module. Fig. 8.9 shown platform screenshot during heterogeneous page fragmentation.

The fragments are displayed on the screen (Fig. 8.10). There is also a file with the coordinates of fragments. Fragmentation can be corrected manually. The type of fragment is selected from a drop-down list: text, notes, mathematical formulas, chemical formulas and structures, chess diagrams. Other fragments are assigned the “image” type by default. Fig. 8.12 on p. 147 shows another available view on fragments, namely, ribbon of fragments.

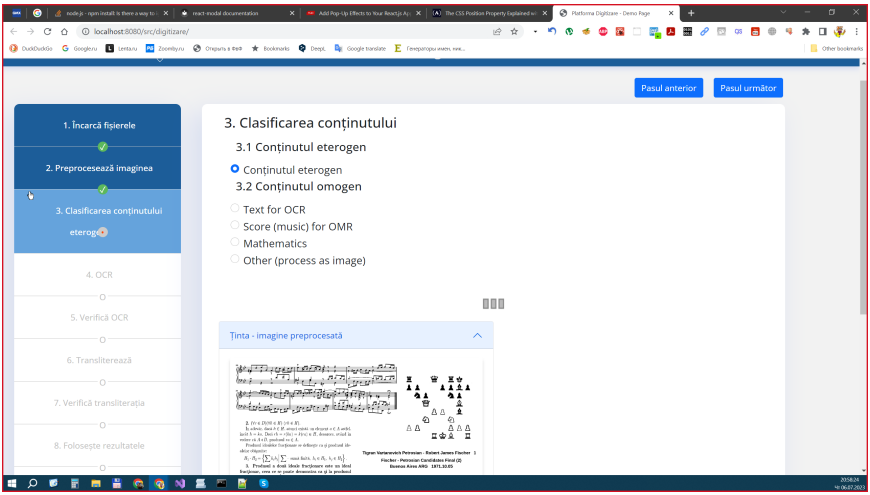


Figure 8.9: Platform HeDy during fragmentation of a heterogeneous page.

Specialized programs (FineReader, Mathpix, etc.) are used to recognize fragments by content type. The result will be a text (see Fig. 8.11) or script representation of the fragment: natural language text, LaTeX script for math calculations, MusicXML for notes, etc. Recognition results are displayed and can be checked and corrected. In some cases you may need to transliterate the recognized text, e.g. from the old Romanian Cyrillic alphabet to the modern Latin alphabet. A special AAconv application is called for this task.

The recognition results, together with images of the original pages, are collected in a PDF file. The PDF file may contain other files as attachments. All recognition results are attached to each page. Also attached are page maps containing the coordinates of each fragment on the page.

The assembled file, which combines the original images with the recognition results and page maps, is available to the user and can be used for a variety of purposes, such as recreating a graphical representation of the original document.

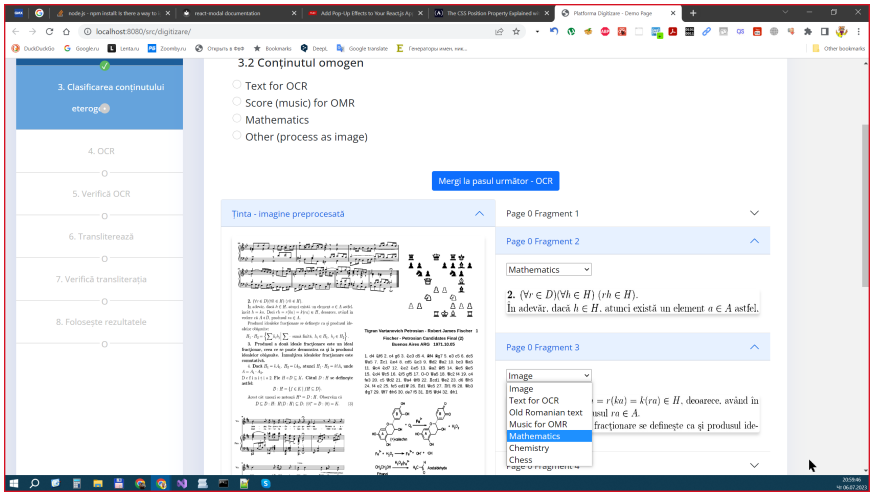


Figure 8.10: Platform HeDy: manual selection of types for page fragments.

8.10 Recognition of mathematical texts

In 2017 we republished the book [230] in the modern Romanian Latin script. Then there were no available tools to recognize formulas, and we had to retype all formulas in LaTeX manually by a highly qualified mathematician. It results in the time consumption shown in Tab. 8.3. Proportion formulas/text is 50/50.

Table 8.3: Manual retype of formulas in 2017.

Work mode	Work	Time
Automated	Scan text, OCR, transliteration	3 man-hours
Manual	Text editing	2 man-days
Manual	Retype of formulas in LaTeX	2 man-months

In just 4 years, the situation has drastically changed. With the advent of deep learning, in 2021 formula recognition has become a common topic of student thesis, and Github is full of them. A number of commercial systems have emerged that recognize complex handwritten and printed formulas.

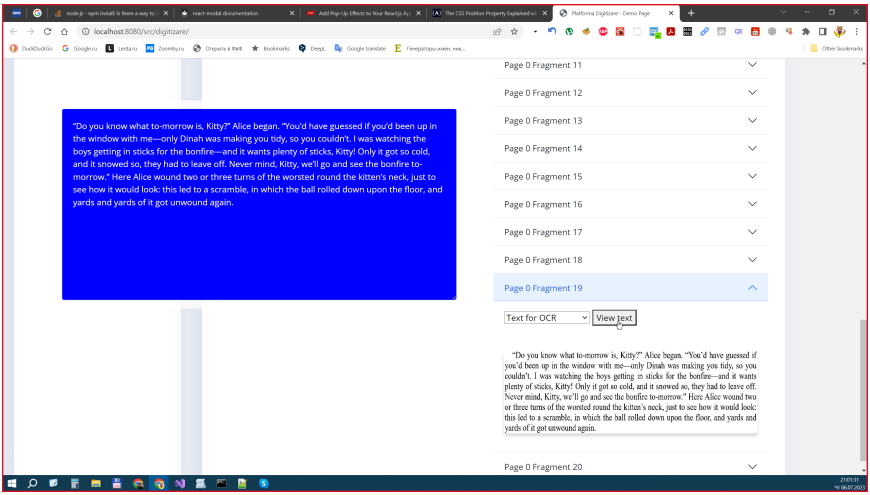


Figure 8.11: Recognized textual fragment in HeDy.

We tested several several open source systems: im2latex [231], image2latex [232], [233], SESHAT [234]. Of these, LaTeX-OCR seemed the most suitable being written in Python, and completed with installation and startup tutorials and with a test set of 200,000 sample formulas that can be expanded by users. However, follow-up testing of LaTeX-OCR revealed its extremely high hardware requirements. To get the best result, we needed to perform a full training on all 200,000 available samples. We failed to run this even after hardware upgrade.

As to the commercial systems, many have only limited capabilities, for example, InftyReader [156]. Here we should note Mathpix [185] that demonstrated the best results. It recognizes complex formulas and texts in many languages, printed and handwritten. Mathpix exports results in LaTeX and has a graphical shell. We tested Mathpix on a page [230, p. 110], and it made only two errors because of scan quality This result is achieved on a page with a language unsupported directly by Mathpix, namely, Romanian in the Cyrillic script. Fig. 8.13-8.14 on pp. 148-148 below demonstrate the processing by Mathpix of the said page. The errors are: 1) absent $ signs around the `\beta` command for the letter β; 2) the letter **Ы** in the word **Ынсэ**

was erroneously recognized as **Ь**.

We can conclude that commercial Mathpix gets the best results in recognition of printed formulas with text. Commercial formula recognizers are implemented as web-based frameworks while open source tools are standalone modules. Deep learning techniques give brilliant results but are extremely dependent on both hardware and software. For example, LaTeX-OCR requires high version of modules and libraries; these high versions in their turn require high version of hardware.

8.11 Digitized Document Management Modules

A digitized document refers to original and pre-processed images, recognized and transliterated texts.

The module uploads pre-processed images. The user can download the images to their personal device for further use. The format of downloaded images is JPG.

The modules for downloading recognized and transliterated text are similar. Downloaded file formats include TXT and DOCX. The DOCX variant is nothing but a raw text shell (without formatting) without preserving the styles or illustrations of the original document.

An important module is storing the digitized document in the platform's database. In addition to storing texts and file references, the digitized object is also saved, which is an object. The object includes preprocessing parameters, recognition and transliteration parameters, recognized and editable text, transliterated and editable text, etc.

To cope with the large number of images uploaded to the platform, we included a module in G4 to upload original documents and pre-processed images to the Cloud . In developing this module, I created a cart , using Amazon S3 (Simple Storage Service), a cloud storage service developed by Amazon Web Services (AWS). Amazon S3 shopping cart is a container for storing files in Amazon S3. We can store any type of files in the Recycle Bin : from images, music and videos to apps, websites, etc. The Recycle Bin is identified by its unique name in Amazon S3 and can be accessed by a URL starting with `https://s3.amazonaws.com/`. It can be customized with

different security settings to protect content and can be used to distribute content through Amazon CloudFront, a content distribution network (CDN).

A set of very important modules in G4 deals with publishing a digitized document. Realizing the existence of the copyright problem and choosing to solve it according to regulations, in the following we will only propose technical solutions that can work with the presumption of solving the legal aspects. The publishing module is centered on the eMoldova portal, in particular based on a portlet called Tezaurul Național Digital (hereafter *Digi*). This portlet contains resources from Moldova's digital treasure trove, as well as some simplified services for digitization and management of digitized documents (so-called Digi articles). Digi includes special working groups to provide the general public with digital articles of the highest quality in terms of the correctness of the recognized/transliterated text. The working groups are divided into two. There is a working group of *digitizers*, those who upload the digitized document to the platform, and a working group of *editors*, those who check and approve the digital article for final publication.

The digitization platform integrates a set of external programs for image preparation, recognition and processing, transliteration and other tasks.

The web application is written in Django, which allows for the development of new modules in Python and for the integration of external programs via Python.

All necessary platform operations are performed by an integrated external application such as Imagemagick, ScanTailor, GIMP, FineReader, etc.

We use a special convergent application integration technique [7]. We will discuss one of the image set processing applications available on the platform.

The page layout is repeated in documents of the same origin, e.g., journal collections for several years. Each book, magazine, or newspaper uses only a limited number of page layouts. These are the design of even and odd pages, beginning and end of chapters, table of contents, etc. About the revitalization of books, this means that the page fragments are placed in approximately the same place. The page layout is repeated in serial books as well.

A subsystem based on a deep learning neural network trained to recognize layout pages is implemented. This is followed by fragmentation of pages

according to their composition. This greatly facilitates further extraction of sub-fragments with homogeneous content, since the possible content of each element of a given layout is more or less fixed.

Additionally, the following tasks were solved: accumulation of files on disk and session management.

File management. We have a client-server program. The client is in JavaScript/Django, the server is in Python. When running, files are uploaded to the server, processed, and the resulting files are generated and accumulated on the server. We considered the following options for managing the resulting files on the server:

- Cleaning up old files: Regularly check and delete old and unused files. To do this, a task is created in Django that will periodically check the creation date of files and delete files that need to be deleted.
- Once the files are processed on the server, they can be uploaded to cloud storage and references to them stored in the database.
- Limit storage size: Set the maximum size for file storage on the server. When the size reaches the limit, notify the client by prompting them to upload the files to external storage.

Session Management. Since the client is written in JavaScript/Django, session management from Django: Django provides a built-in session management system that allows storing and retrieving session data for each client. Sessions in Django are implemented using cookies or session IDs in URLs. When using session IDs in URLs, the session ID is included in the URL of each page. Django automatically extracts the session ID from the URL and associates it with the session data. In Django code, you can access session data, add and remove values from a session using the request.session object. You can perform the following operations on session data.

- Retrieve a stored value using the key with which it was stored. Values can be of any type, such as strings, numbers, lists, and dictionaries.
- Get the unique identifier of the current session. The identifier associates the session data on the server with a particular client.
- Check for the presence of a key in the session.
- Get all the keys of the session.
- Remove a value with the given key from the session.

- Clear session: remove all values from the session.

Access to a session is done through a subroutine that requires a file name as an argument. The corresponding file name can be obtained by accessing the database using the session ID or another unique identifier associated with the user's current session. A Django model can be created that represents uploaded files and associates them with a session ID or user. In the context of Django, a Model is a class that defines the data structure and behavior for working with a database. Models in Django represent tables in the database and provide a convenient way to interact with the data.

A Django model is defined in a models file (models.py) inside a Django application. It inherits from the base class django.db.models.Model and defines fields, relationships to other models, methods, and other attributes.

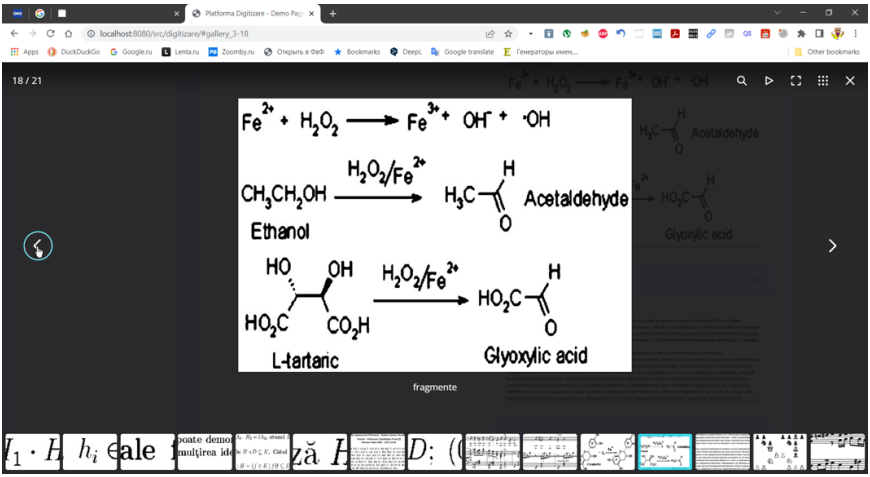


Figure 8.12: Platform HeDy: ribbon of page fragments.

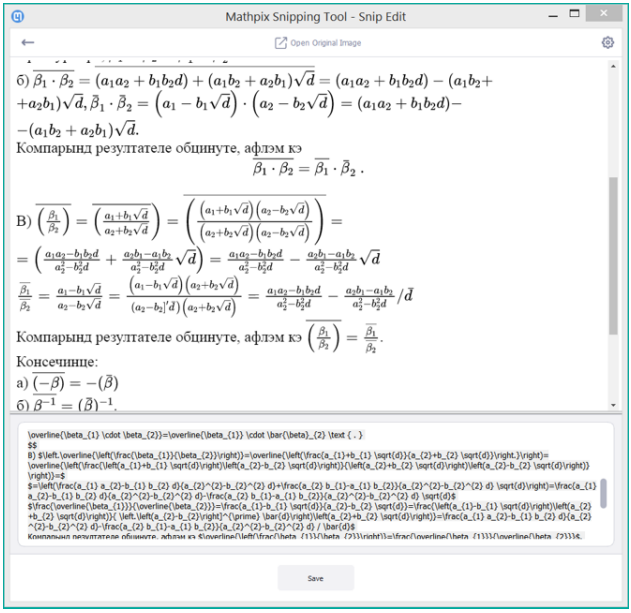


Figure 8.13: Page of a book in Mathpix: source image and resulting LaTeX script.

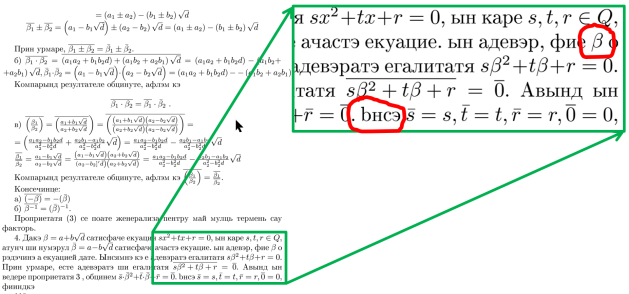


Figure 8.14: Control for errors in Mathpix recognition.

Conclusion

The digitization of historical Romanian documents using our developed technology, as described above, has proven successful, particularly for homogeneous documents. Within our platform, we achieved high accuracy across all processing stages: 95-96

Despite notable achievements, automating the recognition of heterogeneous content remains a challenging problem. The classification of homogeneous fragment types is not entirely resolved, both in academic publications and existing software. Future endeavors may include:

- Enhancing OCR models and transliteration algorithms by incorporating advanced techniques in natural language processing and machine learning to boost accuracy and efficiency.
- Creating a user-friendly interface for bulk OCR model training, broadening accessibility and enabling the construction of practical OCR models for diverse timeframes and printing contexts.
- Addressing font classification issues by grouping fonts into multiple classes, utilizing a combination of convolutional neural network and recurrent neural network architectures to consider the character order in text and improve accuracy.
- Expanding the digitization platform to encompass various document types like manuscripts, maps, etc., thereby broadening access to a diverse range of cultural and historical resources.
- Integrating the digitization platform with other digital tools and resources, such as digital libraries and archives, to facilitate collaboration among researchers and enhance access to additional information.

Moreover, formulating the alignment problem between old and modern

texts requires creating large text corpora encompassing both eras. While these challenges aren't core to our current project, they remain relevant and intriguing for future exploration.

Additional potential tasks involve low-level conversions, such as raster to vector graphics, font extraction from text images, and describing dependent parts of a document. For instance, a table's content visualized as a chart should be accompanied by script files detailing chart parameters. This ensures efficient regeneration and synchronization when the underlying table is modified.

Further potential tasks include font selection and representation in modern formats, analysis and generalization of page structures, grouping identical ornaments, recreating idealized document forms, and the ability to import data from other programs like Excel or Word. Undertaking these tasks implies leveraging Deep Learning methods and conducting in-depth research within new projects.

Bibliography

[1] *Commission recommendation of 27 October 2011 on the digitisation and online accessibility of cultural material and digital preservation.* 2011. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32011H0711 (visited on 12/18/2023).

[2] *Project Gutenberg.* 2023. URL: http://www.gutenberg.org/ (visited on 12/18/2023).

[3] *HathiTrust.* 2023. URL: https://www.hathitrust.org/ (visited on 12/18/2023).

[4] *The Universal Digital Library. Million Book Collection.* 2008. URL: http://ulib.isri.cmu.edu/ (visited on 12/18/2023).

[5] *Google Books.* 2023. URL: https://books.google.com/ (visited on 12/18/2023).

[6] *Europeana. Collections.* 2023. URL: https://www.europeana.eu/en/collections (visited on 12/18/2023).

[7] S. Cojocaru et al. "On convergent technology in development of information systems for processing of documents with heterogeneous content." In: *Proceedings of the Workshop on Intelligent Information Systems.* Chișinău, Republic of Moldova, Dec. 5, 2020, pp. 61–68.

[8] A. Colesnicov, S. Cojocaru, and L. Malahov. "Recognition of heterogeneous documents: problems and challenges." In: *Proceedings of the 5th Conference on Mathematical Foundations of Informatics*. Iași: Alexandru Ion Cuza University Publishers, 2019, pp. 231–245. ISBN: 978-606-714-481-9.

[9] *Tetraevanghelul*. Romanian. Brașov: Coresi, 1561.

[10] Petre P. Panaitescu. *Începuturile şi biruința scrisului în limba română*. Romanian. București: Editura Academiei, 1965. 232 pp.

[11] Pârvu Boerescu. *Din istoria scrierii românești*. Romanian. București: Editura Academiei Române, 2014. 400 pp. ISBN: 978-973-27-2459-0.

[12] Radu Tempea. *Gramatică românească*. Romanian. Sibiu: publisher, 1797. 228 pp.

[13] *Noul Testament*. Romanian. Bălgrad, 1648.

[14] Gheorghe Asachi. *Elemente de matematică. Aritmetică*. Romanian. .1. Iași: Tipografia Albinei, 1836.

[15] Ștefan Ciobanu. *Cultura românească în Basarahia sub stapânirea rusă*. Romanian. Chișinău: Asociatia "Uniunea Culturală Bisericească din Chisinău", 1923. 344 pp.

[16] Adina Dragomirescu. "Ortografia limbii române: definiţie, scurt istoric, instrumente. Principiul fonologic." Romanian. In: *Limba română* XXII.1-2 (2012). ISSN: 0235-9111. URL: https://limbaromana.md/index.php?go=articole&n=1353 (visited on 10/05/2023).

[17] *Dicţionarul ortografic, ortoepic şi morfologic al limbii române*. Romanian. II. Bucureşti: Editura Univers Enciclopedic, 2005.

[18] *Limba română*. URL: https://ro.wikipedia.org/wiki/Limba_rom%C3%A2n%C4%83 (visited on 10/05/2023).

[19] Victor Ufnarovski. *Acvariu matematic*. Romanian. Trans. by S. Guțu. Mica bibliotecă a elevului: matematică, informatică. Chișinău: Editura "Știința", 1988. 240 pp. ISBN: 5-376-00427-9.

[20] *Scan Tailor. An interactive post-processing tool for scanned pages*. 2021. URL: https://scantailor.org/ (visited on 11/17/2023).

[21] *ABBYY FineReader PDF. Specifications for Win*. URL: https://www.abbyy.com/en-eu/finereader/tech-specs/ (visited on 10/28/2023).

[22] Nobuyuki Otsu. "A Threshold Selection Method from Gray-Level Histograms." In: *IEEE Transactions on Systems, Man, and Cybernetics* 9.1 (1979), pp. 62–66. DOI: 10.1109/TSMC.1979.4310076.

[23] S.J. Lu and C.L. Tan. "Binarization of Badly Illuminated Document Images through Shading Estimation and Compensation." In: *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*. IEEE, Sept. 2007. DOI: 10.1109/icdar.2007.4378723.

[24] Abraham Savitzky and Marcel J. E. Golay. "Smoothing and Differentiation of Data by Simplified Least Squares Procedures." In: *Analytical Chemistry* 36.8 (July 1964), pp. 1627–1639. ISSN: 0003-2700. DOI: 10.1021/ac60214a047.

[25] David Doermann and Karl Tombre. *Handbook of document image processing and recognition*. Springer London, 2014. ISBN: 9780857298591. DOI: 10.1007/978-0-85729-859-1.

[26] Michael Piotrowski. *Natural Language Processing for Historical Texts*. Springer International Publishing, 2012. ISBN: 9783031021466. DOI: 10.1007/978-3-031-02146-6.

[27] Carolyn Strange et al. "Mining for the Meanings of a Murder: The Impact of OCR Quality on the Use of Digitized Historical Newspapers."

In: *Digital Humanities Quarterly* 8.1 (2014). URL: http://www.digitalhumanities.org/dhq/vol/8/1/000168/000168.html.

[28] Thorsten Vobl et al. "PoCoTo - an open source system for efficient interactive postcorrection of OCRed historical texts." In: *ACM International Conference Proceeding Series* (May 2014), pp. 57–61. DOI: 10.1145/2595188.2595197.

[29] ABBYY. *How AI Powers PDF Software & Technology Trends*. 2023. URL: https://pdf.abbyy.com/blog/finereader-powered-by-ai/ (visited on 12/10/2023).

[30] U. Springmann and A. Ludeling. "OCR of historical printings with an application to building diachronic corpora: A case study using the RIDGES herbal corpus." In: *Digital Humanities Quarterly 11, 2 (2017)* (Aug. 2016). DOI: 10.48550/ARXIV.1608.02153. arXiv: 1608.02153 [cs.CL].

[31] T. Bumbu et al. "User Interface to Access Old Romanian Documents." In: *Proceedings of the 4th Conference of Mathematical Society of Moldova CMSM4-2017, June 25-July 2*. 2017, pp. 479–482.

[32] Tudor Bumbu. "Towards a Font Classification Model for Romanian Cyrillic Documents." In: *Computer Science Journal of Moldova* 29.3(87) (2021), pp. 291–298.

[33] Thomas M. Breuel et al. "High-Performance OCR for Printed English and Fraktur Using LSTM Networks." In: *2013 12th International Conference on Document Analysis and Recognition*. 2013, pp. 683–687. DOI: 10.1109/ICDAR.2013.140.

[34] *RIDGES - Register in Diachronic German Science*. 2020. URL: http://korpling.german.hu-berlin.de/ridges/index_en.html (visited on 12/10/2023).

[35] *Tesseract*. URL: https://github.com/tesseract-ocr/ (visited on 10/28/2023).

[36] Adam Dudczak, Aleksandra Nowak, and Tomasz Parkoła. "Creation of custom recognition profiles for historical documents." In: *Proceedings of the First International Conference on Digital Access to Textual Cultural Heritage*. DATeCH 2014. ACM, May 2014. DOI: 10.1145/2595188.2595209.

[37] eMOP. *Franken+*. 2023. URL: https://emop.tamu.edu/outcomes/Franken-Plus (visited on 12/10/2023).

[38] Text Creation Partnership. *Eighteenth Century Collections Online*. 2019. URL: https://quod.lib.umich.edu/e/ecco/ (visited on 12/10/2023).

[39] Taylor Berg-Kirkpatrick and Dan Klein. "Improved Typesetting Models for Historical OCR." In: *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Association for Computational Linguistics, 2014. DOI: 10.3115/v1/p14-2020.

[40] Marcin Helinski, Milosz Kmieciak, and Tomasz Parkola. *Report on the comparison of Tesseract and ABBYY FineReader OCR engines*. 2012. URL: https://api.semanticscholar.org/CorpusID:6341954 (visited on 12/14/2023).

[41] ABBYY. *Research Guides: ABBYY FineReader Tutorial: Creating and Training a User Pattern*. 2023. URL: https://guides.nyu.edu/abbyy/training-abbyy (visited on 12/10/2023).

[42] N. White. "Training Tesseract for Ancient Greek OCR." In: *Eutypon* 28-29 (2012), pp. 1–11. eprint: https://ancientgreekocr.org/e29-a01.pdf.

[43] Mamata Nayak and Ajit Kumar. "Odia Characters Recognition by Training Tesseract OCR Engine." In: *International Journal of Computer Applications* (July 2013), pp. 975–8887.

[44] C. Clausner, A. Antonacopoulos, and S. Pletschacher. "Efficient and effective OCR engine training." In: *International Journal on Document Analysis and Recognition* 23.1 (2020), pp. 73–88. ISSN: 1433-2833.

[45] C. Clausner, S. Pletschacher, and A. Antonacopoulos. "Aletheia - An Advanced Document Layout and Text Ground-Truthing System for Production Environments." In: *2011 International Conference on Document Analysis and Recognition*. IEEE, Sept. 2011. DOI: 10.1109/icdar.2011.19.

[46] Stefan Pletschacher and Apostolos Antonacopoulos. "The PAGE (Page Analysis and Ground-truth Elements) format framework." In: Aug. 2010, pp. 257–260. DOI: 10.1109/ICPR.2010.72.

[47] Christian Clausner, Apostolos Antonacopoulos, and Stefan Pletschacher. "ICDAR2019 Competition on Recognition of Documents with Complex Layouts - RDCL2019." In: *2019 International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, Sept. 2019. DOI: 10.1109/icdar.2019.00245.

[48] *IMPACT Centre of Competence — digitisation.eu*. 2023. URL: https://www.digitisation.eu/ (visited on 12/10/2023).

[49] *Library: Archival Skills: Transcription — libguides.hull.ac.uk*. 2023. URL: https://libguides.hull.ac.uk/archival-skills/transcription (visited on 12/10/2023).

[50] PRImA. *Text Evaluation tool*. 2023. URL: http://www.primaresearch.org/tools/ (visited on 12/10/2023).

[51] Stephen Vincent Rice. "Measuring the accuracy of page-reading systems." PhD thesis. 1996. DOI: 10.25669/HFA8-0CQV.

[52] *OCRopus*. URL: https://github.com/ocropus (visited on 12/16/2023).

[53] Mittagessen. *Kraken: OCR engine for all the languages*. 2023. URL: https://github.com/mittagessen/kraken (visited on 12/10/2023).

[54] *Calamari-OCR: Line based ATR Engine based on OCRopy*. 2023. URL: https://github.com/Calamari-OCR/calamari (visited on 12/10/2023).

[55] M. V. Valueva et al.
"Application of the residue number system to reduce hardware costs of the convolutional neural network implementation."
In: *Mathematics and Computers in Simulation* 177 (2020), pp. 232–243. ISSN: 0378-4754.
DOI: https://doi.org/10.1016/j.matcom.2020.04.031.

[56] Uwe Springmann et al. "OCR of historical printings of Latin texts: problems, prospects, progress."
In: *Proceedings of the First International Conference on Digital Access to Textual Cultural Heritage*. 2014, pp. 71–75.
DOI: 10.1145/2595188.2595205.

[57] Faisal Shafait. "Document image analysis with OCRopus."
In: *2009 IEEE 13th International Multitopic Conference* (2009), pp. 1–6.

[58] Christoph Wick, Christian Reul, and Frank Puppe.
"Calamari - A High-Performance Tensorflow-based Deep Learning Package for Optical Character Recognition."
In: *Digital Humanities Quarterly 14 (2), 2020* (July 2018).

[59] David M. Allen. "The Relationship Between Variable Selection and Data Agumentation and a Method for Prediction."
In: *Technometrics* 16.1 (Feb. 1974), pp. 125–127. ISSN: 1537-2723.
DOI: 10.1080/00401706.1974.10489157.

[60] M. Stone.
"Cross-Validatory Choice and Assessment of Statistical Predictions."
In: *Journal of the Royal Statistical Society: Series B (Methodological)* 36.2 (Jan. 1974), pp. 111–133. ISSN: 2517-6161.
DOI: 10.1111/j.2517-6161.1974.tb00994.x.

[61] R. Christensen. *Thoughts on prediction and cross-validation.* 2015. URL: https://math.unm.edu/~fletcher/Prediction.pdf (visited on 12/10/2023).

[62] Senka Drobac and Krister Linden. "Optical character recognition with neural networks and post-correction with finite state methods." In: *International Journal on Document Analysis and Recognition (IJDAR)* 23.4 (Aug. 2020), pp. 279–295. ISSN: 1433-2825. DOI: 10.1007/s10032-020-00359-9.

[63] Christoph Wick, Christian Reul, and Frank Puppe. "Comparison of OCR Accuracy on Early Printed Books using the Open Source Engines Calamari and OCRopus." In: *Journal for Language Technology and Computational Linguistics* 33.1 (July 2018), pp. 79–96. ISSN: 2190-6858. DOI: 10.21248/jlcl.33.2018.219.

[64] *Digital Materials of Finland: The newspaper collection.* 2023. URL: https://digi.kansalliskirjasto.fi/search?formats=NEWSPAPER (visited on 12/10/2023).

[65] P. Kauppinen. *OCR Post-Processing by Parallel Replace Rules Implemented as Weighted Finite-State Transducers. Master's thesis.* 2016. URL: https://helda.helsinki.fi/handle/10138/162866 (visited on 12/10/2023).

[66] Senka Drobac, Pekka Kauppinen, and Krister Lindén. "Improving OCR of historical newspapers and journals published in Finland." In: *Proceedings of the 3rd International Conference on Digital Access to Textual Cultural Heritage.* DATeCH2019. ACM, May 2019. DOI: 10.1145/3322905.3322914.

[67] Standards - Library of Congress. *ALTO: Technical Metadata for Optical Character Recognition.* 2023. URL: https://www.loc.gov/standards/alto/techcenter/use-with-mets.html (visited on 12/10/2023).

[68] Vladimir I. Levenshtein. "Binary codes capable of correcting deletions, insertions, and reversals." In: *Soviet physics. Doklady* 10 (1965), pp. 707–710.

[69] Ladislav Lenc et al.
"HDPA: historical document processing and analysis framework."
In: *Evolving Systems* 12.1 (May 20, 2020), pp. 177–190.
ISSN: 1868-6478. DOI: 10.1007/s12530-020-09343-4.

[70] *OCR Corpora and Tools*. 2022.
URL: http://ocr-corpus.kiv.zcu.cz (visited on 12/10/2023).

[71] *Django — djangoproject.com*. 2023.
URL: https://www.djangoproject.com (visited on 12/10/2023).

[72] J. Sauvola and M. Pietikäinen.
"Adaptive document image binarization."
In: *Pattern Recognition* 33.2 (Feb. 2000), pp. 225–236. ISSN: 0031-3203.
DOI: 10.1016/s0031-3203(99)00055-2.

[73] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation." In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Springer International Publishing, 2015, pp. 234–241.
ISBN: 9783319245744. DOI: 10.1007/978-3-319-24574-4_28.

[74] Christian Clausner et al.
"The ENP image and ground truth dataset of historical newspapers."
In: *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, Aug. 2015.
DOI: 10.1109/icdar.2015.7333898.

[75] *Bavarian-Czech network of digital historical sources*. 2023.
URL: https://www.portafontium.eu (visited on 12/10/2023).

[76] Baoguang Shi, Xiang Bai, and Cong Yao. "An End-to-End Trainable Neural Network for Image-Based Sequence Recognition and Its Application to Scene Text Recognition."
In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.11 (Nov. 2017), pp. 2298–2304. ISSN: 2160-9292.
DOI: 10.1109/tpami.2016.2646371.

[77] Henry S. Baird. "Anatomy of a versatile page reader."
In: *Proceedings of the IEEE* 80.7 (July 1992), pp. 1059–1065.
ISSN: 1558-2256. DOI: 10.1109/5.156469.

[78] Roldano Cattoni et al. "Geometric Layout Analysis Techniques for Document Image Understanding: a Review." In: (Nov. 2000).

[79] Tobias Strauss et al. "System Description of CITlab's Recognition & Retrieval Engine for ICDAR2017 Competition on Information Extraction in Historical Handwritten Records." In: (Apr. 2018).
DOI: 10.48550/ARXIV.1804.09943. arXiv: 1804.09943.

[80] readcoop. *Transkribus project*. 2023.
URL: https://readcoop.eu/transkribus (visited on 12/10/2023).

[81] *OCR-D — DFG-funded Initiative for Optical Character Recognition Development*. 2023.
URL: https://readcoop.eu/transkribus (visited on 12/10/2023).

[82] Christian Reul et al. "OCR4all – An Open-Source Tool Providing a (Semi-)Automatic OCR Workflow for Historical Printings."
In: (Sept. 9, 2019). ISSN: 2076-3417. arXiv: 1909.04032.

[83] Yulia Chernyshova, Alexander Gayer, and Alexander Sheshkus. "Generation method of synthetic training data for mobile OCR system."
In: *Tenth International Conference on Machine Vision (ICMV 2017)*. Ed. by Jianhong Zhou et al. SPIE, Apr. 2018.
DOI: 10.1117/12.2310119.

[84] V. Margner and M. Pechwitz.
"Synthetic data for Arabic OCR system development."
In: *Proceedings of Sixth International Conference on Document Analysis and Recognition*. ICDAR-01. IEEE Comput. Soc, 2001, pp. 1159–1163. DOI: 10.1109/icdar.2001.953967.

[85] L. Malahov et al. "Optical Character Recognition Applied to Romanian Printed Texts of the 18th-20th Century."
In: *Computer Science Journal of Moldova* 24.1 (2016), pp. 106–117.

[86] DeLORo Team.
Deep Learning for Old Romanian — Academia Romana. 2023. URL: http://deloro.iit.academiaromana-is.ro (visited on 12/10/2023).

[87] Padurariu C. and D. Cristea. "Solution for scanned documents segmentation and letter recognition." In: *Proceedings of the 14th edition of the International Conference on Linguistic Resources and Tools for Natural Language Processing – ConsILR-2019*. 2019, pp. 127–137.

[88] D. Cristea et al.
"From Scan to Text. Methodology, Solutions, and Perspectives of Deciphering Old Cyrillic Romanian Documents into the Latin Script."
In: *Knowledge, Language, Models*.
INCOMA Ltd., Shoumen, Bulgaria, 2020, pp. 38–56.

[89] D. Cristea et al. "Data Structure and Acquisition in DeLORo – a Technology for Deciphering Old Cyrillic-Romanian Documents."
In: *Proceedings of ConsILR*. 2021, pp. 59–74.

[90] *Colectia Monumenta Linguae Dacoromanorum - Editura Universitii "Alexandru Ioan Cuza" din Iasi,* 2015.
URL: https://www.editura.uaic.ro/produse/colectii/monumenta_linguae_dacoromanorum/1 (visited on 12/10/2023).

[91] Ludmila Malahov, Catalina Maranduc, and Alexandru Colesnicov.
"A Diachronic Corpus for Romanian (RoDia)."
In: *Proceedings of the First Workshop on Language technology for Digital Humanities in Central and (South-)Eastern Europe*.
Varna: INCOMA Inc., Sept. 2017, pp. 1–9.

[92] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le.
"Sequence to Sequence Learning with Neural Networks." In: (2014).
arXiv: 1409.3215.

[93] Radu Tudor Ionescu, Marius Popescu, and Aoife Cahill.
"String Kernels for Native Language Identification: Insights from Behind the Curtains."
In: *Computational Linguistics* 42.3 (Sept. 2016), pp. 491–525.
ISSN: 1530-9312. DOI: 10.1162/coli_a_00256.

[94] Ery Arias-Castro, Guangliang Chen, and Gilad Lerman.
"Spectral clustering based on local linear approximations."
In: *Electronic Journal of Statistics* 5.none (Jan. 2011). ISSN: 1935-7524.
DOI: 10.1214/11-ejs651.

[95] Steffen Eger, Tim vor der Brück, and Alexander Mehler.
"A Comparison of Four Character-Level String-to-String Translation Models for (OCR) Spelling Error Correction." In: *The Prague Bulletin of Mathematical Linguistics* 105.1 (Apr. 2016), pp. 77–99.
ISSN: 1804-0462. DOI: 10.1515/pralin-2016-0004.

[96] Rafael Llobet et al.
"OCR Post-processing Using Weighted Finite-State Transducers."
In: *20th International Conference on Pattern Recognition, ICPR 2010, Istanbul, Turkey, 23-26 August 2010*, pp. 2021–2024.

[97] Jorge Ramon Fonseca Cacho.
"Improving OCR Post Processing with Machine Learning Tools."
PhD thesis. 2019. DOI: 10.34917/16076262.

[98] Christian Reul et al. "Improving OCR Accuracy on Early Printed Books by utilizing Cross Fold Training and Voting."
In: (Nov. 27, 2017). arXiv: 1711.09670.

[99] Miikka Silfverberg, Pekka Kauppinen, and Krister Linden.
"Data-Driven Spelling Correction using Weighted Finite-State Methods." In: *Proceedings of the SIGFSM Workshop on Statistical NLP and Weighted Automata*.
Association for Computational Linguistics, 2016, pp. 51–59.

[100] Michel Genereux et al.
"Correcting OCR errors for German in Fraktur font."
In: *Proceedings of the First Italian Conference on Computational Linguistics (CLiC-it 2014)*. 2014, pp. 186–190.

[101] Grigori Sidorov et al.
"Syntactic Dependency-Based N-grams as Classification Features." In: *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2013, pp. 1–11. DOI: 10.1007/978-3-642-37798-3_1.

[102] Ido Kissos and Nachum Dershowitz. "OCR Error Correction Using Character Correction and Feature-Based Word Classification." In: *2016 12th IAPR Workshop on Document Analysis Systems (DAS)*. 2016, pp. 198–203. DOI: 10.1109/das.2016.44.

[103] S. Cojocaru et al. "Optical Character Recognition Applied to Romanian Printed Texts of the 18th-20th Century." In: *Computer Science Journal of Moldova* 1(70) (2016), pp. 106–117. ISSN: 1561-4042.

[104] Veronica Romero, Alejandro Hector Toselli Rossi, and Enrique Vidal. "Multimodal Interactive Handwritten Text Transcription." In: *Series in Machine Perception and Artificial Intelligence*. 2012, pp. 63–73.

[105] Tobias Englmeier, Florian Fink, and Klaus Schulz. "A-I-PoCoTo: Combining Automated and Interactive OCR Postcorrection." In: May 2019, pp. 19–24. ISBN: 978-1-4503-7194-0. DOI: 10.1145/3322905.3322908.

[106] Mika Hamalainen and Simon Hengchen. "From the Paft to the Fiiture: a Fully Automatic NMT and Word Embeddings Method for OCR Post-Correction." In: Oct. 2019, pp. 432–437.

[107] Tomas Mikolov et al. "Efficient Estimation of Word Representations in Vector Space." In: (Jan. 2013). arXiv: 1301.3781.

[108] Martin Reynaert. "OCR Post-Correction Evaluation of Early Dutch Books Online - Revisited." In: *International Conference on Language Resources and Evaluation*. 2016, pp. 967–974.

[109] A. Colesnicov et al. "Development of a platform for processing heterogeneous printed documents." In: *Proceedings of the Conference on Mathematical Foundations of Informatics MFOI-2020*. Kyiv, Ukraine, 2021, pp. 108–122. ISBN: 978-966-999-143-0.

[110] *I.R.I.S. A Canon Company*. IRIS S.A. 2023. URL: https://iriscorporate.com/ (visited on 11/17/2023).

[111] *Kofax Intelligent Automation for Digital Workflow Transformation*. 2023. URL: https://www.kofax.com/products/intelligent-automation-platform (visited on 11/17/2023).

[112] Nishant Subramani et al. "A Survey of Deep Learning Approaches for OCR and Document Understanding." In: (Nov. 2020). DOI: 10.48550/ARXIV.2011.13534. arXiv: 2011.13534 [cs.CL].

[113] Velibor Ilić et al. *SCyDia – OCR for Serbian Cyrillic with Diacritics*. 2022. URL: https://www.academia.edu/88712331/SCyDia_OCR_for_Serbian_Cyrillic_with_Diacritics (visited on 12/04/2023).

[114] Xingjiao Wu et al. "Document image layout analysis via explicit edge embedding network." In: *Information Sciences* 577 (Oct. 2021), pp. 436–448. DOI: 10.1016/j.ins.2021.07.020.

[115] Jaya Krishna Mandivarapu et al. "Efficient Document Image Classification Using Region-Based Graph Neural Network." In: (June 2021). DOI: 10.48550/ARXIV.2106.13802. arXiv: 2106.13802 [cs.CV].

[116] Goutham Kallempudi et al. "Toward Semi-Supervised Graphical Object Detection in Document Images." In: *Future Internet* 14.6 (June 2022), p. 176. DOI: 10.3390/fi14060176.

[117] Souhail Bakkali et al. "Visual and Textual Deep Feature Fusion for Document Image Classification." In: *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, June 2020, pp. 2394–2403. DOI: 10.1109/cvprw50498.2020.00289.

[118] Mouli Rastogi et al. "Information Extraction from Document Images via FCA based Template Detection and Knowledge Graph Rule Induction." In: *2020 IEEE/CVF Conference on Computer Vision and Pattern*

Recognition Workshops (CVPRW). IEEE, June 2020, pp. 2377–2385. DOI: 10.1109/cvprw50498.2020.00287.

[119] Frieda Josi, Christian Wartena, and Ulrich Heid. "Preparing Legal Documents for NLP Analysis: Improving the Classification of Text Elements by Using Page Features." In: (2022), pp. 17–29. DOI: 10.25968/OPUS-2161.

[120] Francesco Visalli, Antonio Patrizio, and Massimo Ruffolo. "A Two Step Fine-tuning Approach for Text Recognition on Identity Documents." In: *Proceedings of the 13th International Conference on Agents and Artificial Intelligence*. SCITEPRESS - Science and Technology Publications, Jan. 2021, pp. 837–844. DOI: 10.5220/0010252208370844.

[121] L. Malahov et al. "On text processing after OCR." In: *Proceedings of 13-th International Conference on Linguistic Resources and Tools for Processing Romanian Language, Iasi, 22 – 23 November 2018*. 2018, pp. 53–60.

[122] Amfilohie Hotiniul. *De obște gheografie*. Romanian. 1795. URL: https://dspace.bcucluj.ro/bitstream/123456789/67742/1/BCUCLUJ_FCS_BRV594.pdf (visited on 12/15/2023).

[123] Ștefan Cazimir. *Alfabetul de tranziție ; Jurnal de tranziție*. Oscar Print, 1996. 197 pp. ISBN: 9789739757348.

[124] *Tipărituri vechi [Old printings]*. Romanian. 2023. URL: https://tipariturиromanesti.wordpress.com/ (visited on 12/16/2023).

[125] Vasile Alecsandri. "Suvenire din Italia. Buchetiera de la Florența." Romanian. In: *Dacia literară* 1 (1840). Ed. by Mihail Kogălniceanu, pp. 355–407. URL: http://dspace.bcu-iasi.ro/bitstream/handle/123456789/813/BCUIASI_PER_X-1640_1840%2c%20Tom.1.pdf (visited on 12/16/2023).

[126] Vasile Alecsandri. *Buchetiera de la Florența*. Romanian. 1840. URL: https://www.vasilealecsandri.eu/opere/nuvele/buchetiera_de_la_florenta.html (visited on 12/17/2023).

[127] Stefan Andronic. *Radu VII de la Afumați*. 1846. URL: https://revistatransilvania.ro/wp-content/uploads/2019/11/1846.-S.-Andronic-Radu-VII-de-la-Afumati.pdf (visited on 11/27/2023).

[128] *Magazi Istoric pentru Dacia*. Romanian. 1845. URL: https://archive.org/download/magazinuistoric01unkngoog/magazinuistoric01unkngoog_tif.zip (visited on 12/14/2023).

[129] G. Baiculescu and Academia Republicii Socialiste România. Biblioteca. *Publicațiile periodice românești (ziare, gazete, reviste).: Catalog alfabetic: 1919-1924*. Publicațiile periodice românești. Editura Academiei Republicii Socialiste România, 1987. URL: https://books.google.md/books?id=chkzAQAAIAAJ.

[130] Tiberiu Boros and Adrian Zafiu. "Transliterare automata din engleza în romana. Aplicatii si rezultate." In: *Romanian Journal of Human - Computer Interaction* 5.3 (2012), pp. 1–14.

[131] Min Zhang, Haizhou Li, and Jian Su. "Direct Orthographical Mapping for Machine Transliteration." In: *International Conference on Computational Linguistics*. 2004, pp. 716–722.

[132] Marius Mioc. *31 August 1989 — adoptarea legilor despre limba de stat şi grafia latină în RSS Moldovenească*. 2014. URL: https://mariusmioc.wordpress.com/2009/08/30/31-august-1989/ (visited on 12/10/2023).

[133] Ioana Vintilă-Rădulescu. *DIN – Dicționar normativ al limbii române: ortografic, ortoepic, morfologic și practic*. București: Editura Corint, 2009. ISBN: 9789731353937.

[134] E. Boian et al. "Digitizarea, recunoasterea si conservarea patrimoniului cultural-istoric." In: *Akademos* 1(32) (2014), pp. 61–68.

[135] T. Erjavec. "MULTEXT-East Version 3: Multilingual Morphosyntactic Specifications, Lexicons and Corpora."
In: *International Conference on Language Resources and Evaluation*. 2004.

[136] R. Simionescu. "Hybrid POS Tagger."
In: *International Conference on Language Resources and Evaluation*. 2011.

[137] Z. Ziran et al. "Text alignment in early printed books combining deep learning and dynamic programming."
In: *Pattern Recognition Letters* 133 (2020), pp. 109–115.

[138] W. Qin, R. I. Elanwar, and M. Betke.
"LABA: logical layout analysis of book page images in Arabic using multiple support vector machines."
In: *IEEE 2nd international workshop on Arabic and derived script analysis and recognition (ASAR)*. 2018, pp. 35–40.

[139] S. Chadha, S. Mittal, and V. Singhal. "An insight of script text extraction performance using machine learning techniques."
In: *International Journal of Innovative Technology and Exploring Engineering (IJITEE)* 9 (1 Nov. 2019). ISSN: 2278-3075.

[140] Michele Alberti et al.
"Open evaluation tool for layout analysis of document images."
In: *14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*. Kyoto, Nov. 23, 2017, pp. 43–47.

[141] P. Li, X. Jiang, and H. Shatkay.
"Extracting figures and captions from scientific publications."
In: *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*. 2018.

[142] C. Clausner and A. Antonacopoulos. "Ontology and framework for semantic labelling of document data and software methods." In: *13th IAPR International Workshop on Document Analysis Systems (DAS)*. Vienna, 2018, pp. 73–78.

[143] L. Ma et al. “Segmentation and recognition for historical Tibetan document images.” In: *IEEE Access* 8 (2020), pp. 52641–52651. URL: https://ieeeaccess.ieee.org/ (visited on 10/25/2023).

[144] Noah Siegel et al. “Extracting scientific figures with distantly supervised neural networks.” In: *Proceedings of the 18th ACM/IEEE on Joint Conference on Digital Libraries*. May 2018, pp. 223–232. DOI: 10.1145/3197026.3197040.

[145] H. M. Al-Barhamtoshy and A. S. Alghamdi. “A comprehensive framework for OCR Web services system for Arabic calligraphy documents.” In: *International Journal of Engineering and Technology* 8.1.11 (2019). DOI: 10.14419/ijet.v8i1.11.28084.

[146] A. Colesnicov et al. “Semi-automated workow for recognition of printed documents with heterogeneous content.” In: *Computer Science Journal of Moldova* 28.3 (2020), pp. 223–240.

[147] A Colesnicov et al. “On XML Standards to Present Heterogeneous Data and Documents.” In: *Proceedings of Workshop on Intelligent Information Systems WIIS2021*. Chisinau, Republic of Moldova, 2021, pp. 112–117.

[148] Frank D. Julca-Aguilar, Ana L.L.M. Maia, and Nina S.T. Hirata. “Text/Non-Text Classification of Connected Components in Document Images.” In: *2017 30th SIBGRAPI Conference on Graphics, Patterns and Images (SIBGRAPI)*. 2017, pp. 450–455. DOI: 10.1109/SIBGRAPI.2017.66.

[149] Marina Polyakova et al. “Combined method for scanned documents images segmentation using sequential extraction of regions.” In: *Eastern-European Journal of Enterprise Technologies* 5.2 (95) (Sept. 2018), pp. 6–15. ISSN: 1729-3774. DOI: 10.15587/1729-4061.2018.142735.

[150] Sheikh Faisal Rashid et al. “Table Recognition in Heterogeneous Documents Using Machine Learning.” In: *2017 14th IAPR International Conference on Document Analysis*

and Recognition (ICDAR). IEEE, Nov. 2017, pp. 777–782. DOI: 10.1109/icdar.2017.132.

[151] Joshua Staker et al. "Molecular Structure Extraction From Documents Using Deep Learning." In: (Feb. 2018). DOI: 10.48550/ARXIV.1802.04903. arXiv: 1802.04903 [cs.LG].

[152] Chixiang Ma et al. "Robust Table Detection and Structure Recognition from Heterogeneous Document Images." In: *Pattern Recognition* 133 (Jan. 2022), p. 109006. DOI: 10.1016/j.patcog.2022.109006. eprint: 2203.09056.

[153] Weihong Lin et al. "TSRFormer: Table Structure Recognition with Transformers." In: (Aug. 2022). DOI: 10.48550/ARXIV.2208.04921. arXiv: 2208.04921 [cs.CV].

[154] *CIB OCR*. URL: https://doxiview.com/doxiview-home/ (visited on 10/28/2023).

[155] *OCRSpace. Free OCR API and Online OCR*. URL: https://ocr.space/ (visited on 10/28/2023).

[156] *InftyReader 3.1. Performs OCR scanning of scientific documents*. URL: https://inftyreader.software.informer.com/ (visited on 12/07/2023).

[157] *Readiris. Readiris 17, the advanced OCR solution for Windows and Mac*. URL: https://iriscorporate.com/softwares/readiris-17/ (visited on 10/28/2023).

[158] *TopOCR*. URL: https://www.topocr.com/ (visited on 10/28/2023).

[159] *Nebo. Where ideas take shape*. URL: https://www.nebo.app/ (visited on 10/28/2023).

[160] S. Cojocaru et al. "Digitization Technology of Old Romanian Documents Printed in the Cyrillic Script." In: *Horizons in Computer Science Research*. Ed. by Thomas S. Clary. Vol. 21. Nova Science Publishers, Inc., 2021. Chap. 6, pp. 185–217. ISBN: 978-1-68507-677-1.

[161] S. Cojocaru et al. “Instrumentar Multimedia pentru Facilitarea Promovării Arte Muzicale.” Romanian. In: *Studia Universitatis. Științe exacte și economice* 3(13) (2008), pp. 82–85.

[162] Gioachino Rossini. *William Tell Overture. Arranged by Franz Liszt.* For solo piano. 2018.
URL: https://musescore.com/vader915/scores/5203413 (visited on 12/20/2023).

[163] Marcelle Duchesne-Guillemin. *A Hurrian musical score from Ugarit. The discovery of Mesopotamian music.* Undena Publications, 1984. 32 pp.
URL: http://urkesh.org/attach/duchesne-guillermin%201984%20the%20%20discovery%20of%20mesopotamian%20music.pdf (visited on 11/04/2023).

[164] *MIDI Association.* 2023.
URL: https://www.midi.org/ (visited on 11/04/2023).

[165] *musicXML.*
The standard open format for exchanging digital sheet music.
MakeMusic, Inc. 2023.
URL: https://www.musicxml.com/ (visited on 11/04/2023).

[166] *Musescore. Notate your music.* MuseScore Ltd. 2023.
URL: https://musescore.com/ (visited on 11/04/2023).

[167] Gerd Castan. *music-notation.info. Optical Music Recognition (OMR).* 2019.
URL: http://www.music-notation.info/en/compmus/omr.html (visited on 12/19/2023).

[168] A. Colesnicov et al.
“On digitization of documents with script presentable content.”
In: *Proceedings of the 5th Conference on Mathematical Society of the Rep. Moldova IMCS-55.* Sept. 2019, pp. 321–324.
ISBN: 978-9975-68-378-4.

[169] *Visiv SharpEye Music Scanning.* Neuratron Ltd. 2018.
URL: http://www.visiv.co.uk/ (visited on 11/04/2023).

[170] *Mozart. Convert sheet music to a machine-readable version.* 2022. URL: https://github.com/aashrafh/Mozart (visited on 12/19/2023).

[171] *Orchestra.* 2021. URL: https://github.com/AdelRizq/Orchestra (visited on 12/19/2023).

[172] *Audiveris. Audiveris - Open-source Optical Music Recognition.* 2023. URL: https://github.com/Audiveris/audiveris (visited on 12/19/2023).

[173] *Sibelius. Empowering creative composers.* Avid Technology, Inc. 2023. URL: https://www.avid.com/sibelius/ (visited on 11/04/2023).

[174] *Musipedia. The Open Music Encyclopedia.* 2023. URL: https://www.musipedia.org/ (visited on 11/04/2023).

[175] *AnthemScore. Music AI for your PC.* Lunaverus. 2023. URL: https://www.lunaverus.com/ (visited on 11/04/2023).

[176] *CTAN. Comprehensive TEX Archive Network.* URL: https://ctan.org/ (visited on 11/03/2023).

[177] Leslie Lamport. *LATEX: A Document Preparation System.* 2nd ed. Addison-Wesley, 1994. 272 pp. ISBN: 978-0201529838.

[178] *MathML. W3C Math Home.* URL: https://www.w3.org/Math/ (visited on 11/03/2023).

[179] *TikZ.net. Graphics with TikZ in LATEX.* URL: https://tikz.net/ (visited on 11/03/2023).

[180] Stefan Kottwitz. *LATEX Graphics with TikZ. A practitioner's guide to drawing 2D and 3D images, diagrams, charts, and plots.* Packt Publishing, 2023. ISBN: 978-1804618233.

[181] *Maplesoft. The Essential Tool for Mathematics.* URL: https://www.maplesoft.com/products/Maple/ (visited on 11/03/2023).

[182] *Wolfram Mathematica.*
The world's definitive system for modern technical computing. URL: https://www.wolfram.com/mathematica/ (visited on 11/03/2023).

[183] *Maxima. A Computer Algebra System.* Apr. 15, 2022.
URL: http://maxima.sourceforge.io/ (visited on 11/03/2023).

[184] *Maxima Online.*
Web interface is dated back to 2012, contains dead links. 2012.
URL: http://maxima.cesga.es/ (visited on 11/07/2023).

[185] *Mathpix. AI-powered document automation.*
URL: https://mathpix.com/ (visited on 11/03/2023).

[186] L Burtseva et al. "Heterogeneous document processing: Case study of mathematical texts." In: *Abstracts of International Conference Mathematics and It: Research and Education (MITRE-2021) dedicated to the 75th anniversary of Moldova State University.* 2021, pp. 96–97.

[187] *IUPAC. International Union of Pure and Applied Chemistry.*
URL: https://iupac.org/ (visited on 11/03/2023).

[188] *InChI Trust.*
InChI: open-source chemical structure representation algorithm.
URL: https://www.inchi-trust.org/ (visited on 11/03/2023).

[189] *RInChI. The RInChI project.*
URL: https://www-rinchi.ch.cam.ac.uk/ (visited on 11/03/2023).

[190] *CTAN. Chemistry.*
URL: https://ctan.org/topic/chemistry (visited on 11/03/2023).

[191] *SMILES. A Simplified Chemical Language.* URL: http://www.daylight.com/dayhtml/doc/theory/theory.smiles.html (visited on 11/03/2023).

[192] *MDL MOL files. Chemistry.*
URL: https://docs.chemaxon.com/display/docs/MDL+MOLfiles%2C+RGfiles%2C%20+SDfiles%2C+Rxnfiles%2C+RDfiles+formats (visited on 11/03/2023).

[193] *OSRA Wiki.* 2023.
URL: https://sourceforge.net/p/osra/wiki/Home/ (visited on 12/20/2023).

[194] *Imago OCR*. EPAM Systems. 2023. URL: https://lifescience.opensource.epam.com/imago/index.html (visited on 12/20/2023).

[195] *MolVec. Chemical OCR engine*. URL: https://github.com/ncats/molvec (visited on 05/16/2023).

[196] Kohulan. *DECIMER Image Transformer: Deep Learning for Chemical Image Recognition using Efficient-Net V2 + Transformer*. 2023. URL: https://github.com/Kohulan/DECIMER-Image_Transformer (visited on 12/20/2023).

[197] V.K. Ahluwalia. *Stereochemistry of Organic Compounds*. Ane Books Pvt. Limited, 2020. ISBN: 9789388264532.

[198] *Ketcher. Open-source web-based chemical structure editor*. EPAM Systems. 2023. URL: https://lifescience.opensource.epam.com/ketcher/index.html (visited on 11/16/2023).

[199] M.P. Gallaher et al. *Cost Analysis of Inadequate Interoperability in the U.S. Capital Facilities Industry*. Tech. rep. Aug. 2004. DOI: 10.6028/NIST.GCR.04-867.

[200] *Digitization and verification of 2D engineering drawings*. L&T Technology Services Limited. 2023. URL: https://www.ltts.com/whitepaper/digitize-verify-2D-engineering-drawings (visited on 12/19/2023).

[201] *Vectra2D. Built for Engineers and Designers*. Prolincur Technologies LLP. 2022. URL: https://www.prolincur.com/products/vectra2d/ (visited on 12/20/2023).

[202] *Scan2CAD. Convert your files for CAD & CNC*. Avia Systems. 2023. URL: https://www.scan2cad.com/ (visited on 12/20/2023).

[203] *DataSeer. Where AI and Industrial Diagrams Meet*. DataSeer. 2023. URL: https://dataseer.digital/ (visited on 12/20/2023).

[204] *Recognition of handwritten flowcharts with CNNs*. 2023. URL: https://github.com/dbetm/handwritten-flowchart-with-cnn (visited on 12/20/2023).

[205] Aditya Intwala. *Image2CAD*. 2021. URL: https://github.com/adityaintwala/Image2CAD (visited on 12/20/2023).

[206] *ChartReader*. 2021. URL: https://github.com/Cvrane/ChartReader (visited on 12/20/2023).

[207] *Chart Image Classification Using Inception Model*. 2018. URL: https://github.com/arpitjainds/Chart-Image-Classification (visited on 12/20/2023).

[208] *PlotDigitizer*. 2023. URL: https://github.com/dilawar/PlotDigitizer (visited on 12/20/2023).

[209] *Engauge Digitizer*. 2022. URL: https://github.com/markummitchell/engauge-digitizer (visited on 12/20/2023).

[210] Ankit Rohatgi. *WebPlotDigitizer. Extract data from XY charts, Bar graphs, Polar diagrams and much more!* 2023. URL: https://automeris.io/WebPlotDigitizer/ (visited on 12/20/2023).

[211] *PGN-Spec. Standard: Portable Game Notation Specification and Implementation Guide*. Coordinator: Steven J. Edwards. Internet Chess Club. Mar. 12, 1994. URL: https://www.chessclub.com/help/PGN-spec (visited on 11/04/2023).

[212] *CTAN. Search chess*. URL: https://ctan.org/search?phrase=chess.

[213] Gerhard Roth. *ChessOcr. OCR Chess Diagrams*. Android version. Sept. 15, 2015. URL: https://chessocr.en.aptoide.com/ (visited on 11/04/2023).

[214] *ChessAI Buddy*. Chessify, Inc. 2023. URL: https://chessify.me/ (visited on 11/04/2023).

[215] *VRML. Virtual Reality Modeling Language*. 1995.
URL: https://www.w3.org/MarkUp/VRML/ (visited on 11/03/2023).

[216] *Web 3D Consortium.*
Open Standards for Real-Time 3D Communication.
ISO/IEC 19775/19776/19777.
URL: http://www.web3d.org/x3d/what-x3d/ (visited on 11/03/2023).

[217] *HeDy. Platforma de Digitizare*. 2023.
URL: https://github.com/bumbutudor/PlatformaDigitizare
(visited on 11/17/2023).

[218] *Stepper. MUI Core*. Material UI. Version 15.5.0. 2023.
URL: https://mui.com/material-ui/react-stepper/ (visited on 12/17/2023).

[219] Nick Babich. *Wizard Design Pattern*. Apr. 7, 2017. URL:
https://uxplanet.org/wizard-design-pattern-8c86e14f2a38/
(visited on 12/17/2023).

[220] *A->A Conv*. 2019.
URL: https://translitera.cc/ (visited on 12/17/2023).

[221] Markus Diem et al.
"cBAD: ICDAR2017 Competition on Baseline Detection."
In: *2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, 2017, pp. 1355–1360.
ISBN: 978-1-5386-3586-5. DOI: 10.1109/ICDAR.2017.222.

[222] *Welcome to the OpenAI developer platform*. 2023. URL:
https://platform.openai.com/overview (visited on 12/18/2023).

[223] *Models. Overview*. 2023.
URL: https://platform.openai.com/docs/models/overview
(visited on 12/18/2023).

[224] *Amazon S3.*
Object storage built to retrieve any amount of data from anywhere.
Amazon Web Services, Inc. 2023.
URL: https://aws.amazon.com/s3/ (visited on 12/18/2023).

[225] *Amazon CloudFront.*
Securely deliver content with low latency and high transfer speeds.
Amazon Web Services, Inc. 2023.
URL: https://aws.amazon.com/cloudfront/ (visited on 12/18/2023).

[226] *Moldova Digitală. Poarta către tezaurul național.* 2023.
URL: https://emoldova.org/ (visited on 12/18/2023).

[227] T. Bumbu et al. "Discover the Moldovan Cultural Heritage through e-Moldova Portal by Using Crowdsourcing Concept." In: *Proceedings of the Workshop on Intelligent Information Systems WIIS2021.* Chișinău, 2021, pp. 65–75.

[228] *Tezaurul Național Digital.* 2023.
URL: https://digi.emoldova.org/ (visited on 12/18/2023).

[229] *Folclor din părțile Codrilor.* 1973.
URL: https://digi.emoldova.org/d/17-folclor-din-partile-codrilor (visited on 12/18/2023).

[230] V. A. Andrunachievici and I. D. Chitoroagă.
Numere și ideale. Romanian. Reprinted in the Latin script in 2017.
Chișinău: "Lumina", 1979. 224 pp.

[231] *Im2Latex.*
Deep CNN Encoder + LSTM Decoder with Attention for Image to Latex.
URL: https://github.com/luopeixiang/im2latex (visited on 12/07/2023).

[232] *image2latex.* URL: https://github.com/yixuanzhou/image2latex (visited on 12/07/2023).

[233] *LaTeX-OCR.* URL: https://github.com/lukas-blecher/LaTeX-OCR (visited on 12/07/2023).

[234] *SESHAT: Handwritten math expression parser.* 2016.
URL: https://github.com/falvaro/seshat (visited on 12/07/2023).