electronics

MDPI

Article

Software System for Automatic Grading of Paper Tests

Vladimir Jocovic [1,*], Bosko Nikolic [1] and Nebojsa Bacanin [2]

[1] The Department of Computer Science and Information Technology, School of Electrical Engineering, University of Belgrade, Bulevar kralja Aleksandra, 11000 Belgrade, Serbia; nbosko@etf.bg.ac.rs
[2] Faculty of Informatics and Computing, Singidunum University, Danijelova 32, 11000 Belgrade, Serbia; nbacanin@singidunum.ac.rs
* Correspondence: jocke@etf.bg.ac.rs

Abstract: The advent of digital technology has revolutionized numerous aspects of modern life, including the field of assessment and testing. However, paper tests, despite their seemingly archaic nature, continue to hold a prominent position in various assessment domains. The accessibility, familiarity, security, cost-effectiveness, and versatility of paper tests collectively contribute to their continued prominence. Hence, numerous educational institutions responsible for conducting examinations involving a substantial number of candidates continue to rely on paper tests. Consequently, there arises a demand for the possibility of automated assessment of these tests, aiming to alleviate the burden on teaching staff, enhance objectivity in evaluation, and expedite the delivery of test results. Therefore, diverse software systems have been developed, showcasing the capability to automatically score specific question types. Thus, it becomes imperative to categorize related question types systematically, thereby facilitating a preliminary classification based on the content and format of the questions. This classification serves the purpose of enabling effective comparison among existing software solutions. In this research paper, we present the implementation of such a software system using artificial intelligence techniques, progressively expanding its capabilities to evaluate increasingly complex question types, with the ultimate objective of achieving a comprehensive evaluation of all question types encountered in paper-based tests. The system detailed above demonstrated a recognition success rate of 99.89% on a curated dataset consisting of 734,825 multiple-choice answers. For the matching type, it achieved a recognition success rate of 99.91% on 86,450 answers. In the case of short answer type, the system achieved a recognition success rate of 95.40% on 129,675 answers.

Keywords: artificial intelligence; automated test assessment; machine learning; paper test

Citation: Jocovic, V.; Nikolic, B.; Bacanin, N. Software System for Automatic Grading of Paper Tests. *Electronics* **2023**, *12*, 4080. https://doi.org/10.3390/electronics12194080

Academic Editor: Jian Sun

Received: 25 August 2023
Revised: 20 September 2023
Accepted: 23 September 2023
Published: 28 September 2023

1. Introduction

While digital assessments offer advantages such as enhanced interactivity, immediate feedback, and streamlined administration [1], the persistent utilization of paper tests stems from several factors [2]. Firstly, the accessibility of paper tests allows for their deployment in regions with limited access to digital devices or electricity. Secondly, the familiarity associated with the pen-and-paper format engenders a sense of comfort and alleviates test anxiety for many individuals. Thirdly, the perceived security of paper tests cannot be disregarded [3]. Moreover, the cost-effectiveness of paper tests proves advantageous in certain contexts, as they eliminate the need for substantial investments in digital infrastructure and ongoing maintenance [4]. Additionally, the versatility of paper tests in accommodating various assessment formats, particularly those requiring intricate drawings, complex mathematical equations, or freehand writing, remains a prominent advantage [5]. The tangible nature of paper facilitates these tasks more naturally than their digital counterparts [6].

In light of the aforementioned factors, paper tests continue to maintain their widespread usage within educational institutions. However, the reliance on paper tests necessitates a

manual review process, which poses significant challenges, particularly when dealing with a substantial number of candidates. Nonetheless, due to the sheer volume of applicants, the task of evaluation often becomes monotonous, unstimulating, and wearisome. Moreover, the evaluation process introduces a level of subjectivity that can be largely mitigated by utilizing question types that allow for precise and unambiguous responses. Nevertheless, it is inevitable that errors may arise during the evaluation of such a vast number of answer sheets within the review process itself. Hence, there arises a compelling necessity to automate the complete grading process. By doing so, not only would the burden be lifted from reviewers who often find the task uninspiring, but it would also afford them additional time to dedicate to crafting high-quality tests.

Therefore, the imperative to develop a system for automated assessment of paper tests became evident. Initially, automated assessment systems were implemented to handle exclusively multiple-choice questions, with answers separated from the question text and presented on a distinct answer sheet. However, the practice of segregating the questions' and answers' text into separate forms introduces errors in form completion and distracts candidates from the essence of the question itself. Therefore, a desirable feature for the system is the ability to evaluate tests where the questions' and answers' text are unified. Moreover, as the demand arose for automated evaluation of various question types, novel solutions were devised specifically tailored to each respective question type.

Paper tests encompass a diverse array of question types, making it challenging to exhaustively enumerate every potential question variant encountered in these assessments. The presence of synonymous terms that interchangeably denote essentially the same question type further contributes to this complexity. However, for the purpose of facilitating transparent and comparative evaluation of implemented software systems capable of automatically assessing paper tests, it becomes crucial to classify distinct question types and consolidate related question types into cohesive classes.

The following classes have been identified:

- Multiple Choice: Candidates select one or more correct answers from a matrix-like arrangement of choices, typically represented by circles or squares.
- Matching: Candidates establish associations between related concepts or underline specific responses.
- Short Answer: Candidates provide concise answers in the form of one or more words or numerical values, without requiring an extensive interpretation of the written content.
- Essay: Candidates compose extended written responses, emphasizing a comprehensive understanding and articulation of the underlying concepts during the evaluation process.

These question classes are presented in ascending order of complexity, considering the technologies required to effectively address them. Initially, the implemented system focused solely on automating the evaluation of multiple-choice questions [7], and a comprehensive account of this capability can be found in previous research [8]. However, subsequent developments have facilitated the expansion of the system to encompass the automated assessment of matching questions, where correct answers are underlined, as well as short answer questions involving numerical responses. It is envisaged that future iterations of the system will further evolve to encompass evaluation capabilities across all the aforementioned question classes, thereby automating the entire assessment process.

The primary contributions of the presented work can be summarized as follows:

- Examination and classification of the questions employed in the tests.
- Assessment and categorization of current systems designed for automated test grading.
- Development and deployment of an automated test grading system utilizing artificial intelligence techniques, which outperformed other existing tools.
- A comprehensive evaluation of the implemented system across a vast collection of digitally formatted paper tests.

The rest of this paper is organized as follows. Section 2 of this paper presents a collection of software systems that demonstrate the ability to automatically score specific categories of questions. In Section 3, a control flow diagram of the developed software system is presented and illuminated. The subsequent Section 4 highlights the implementation specifics of the system, placing particular emphasis on the components responsible for evaluating the aforementioned question types. Section 5 presents an evaluation of the implemented software system, encompassing performance data, limitations encountered, and challenges faced during its development. Finally, Section 6 concludes the paper, by providing a brief summary and outlining future endeavors and enhancements to be pursued.

2. Related Work

The initial phase involved conducting a comprehensive search for software systems capable of automated grading across the aforementioned question types. Extensive exploration was carried out within the realm of open-access literature. An analysis of their accuracy, performance, and limitations will be undertaken and thoroughly discussed.

It has become apparent that each system is designed to specialize in the evaluation of a particular question class. This specialization has facilitated the identification of notable resemblances among software systems belonging to the same class. Nonetheless, even among software systems capable of automatically scoring the same question class, diverse approaches to detection and grading exist, even when utilizing identical technologies.

The filtering criteria of identified software systems excluded software systems documented in papers published more than a decade ago, ensuring a focus on recent advancements. Meanwhile, the prioritization criteria favored software systems with recent publication dates or those that garnered a higher number of citations, indicative of their prominence in the field.

The chosen systems satisfying the previously given criteria were developed at the School of Electrical Engineering, University of Belgrade (SEE-UB) [8], School of Engineering, Edith Cowan University (SE-ECU) [9], Department of Telematic Engineering, University Carlos III of Madrid (DTE-UCM) [10], School of Electronic and Information Engineering, Foshan University (SEIE-FU) [11], Prince Mohammad bin Fahd University (PMFU) [12], Artificial intelligence Department, Faculty of Computers and Artificial Intelligence, Benha University (FCAI-BU) [13], Information Technologies Division, Adana Alparslan Turkes Science and Technology University (ITD-AATSU) [14], School of Software South China, University of Technology Guangzhou and College of Medical Information Engineering, Guangzhou University of Chinese Medicine (SSSC-UTG/CMIE-GUCM) [15].

Each of the selected software systems will be accompanied by a concise overview of introductory details. This includes the software system's designated name, year of publication, applicable domains, and the corresponding methods used. The selected software systems and their introductory details can be seen in Table 1.

Table 1. Introductory information about selected software systems.

Software System	Year	Question Class	Methods
SEE-UB [8]	2022.	Multiple-choice	Python, Open-CV, CNN
SE-ECU [9]	2018.	Multiple-choice	MATLAB, Custom image processing
DTE-UCM [10]	2013.	Multiple-choice	Python, Open-CV, Custom enhancements
SEIE-FU [11]	2021.	Matching	Python, YOLO
PMFU [12]	2019.	Matching	Python, MATLAB, CNN
FCAI-BU [13]	2022.	Short answer	Python, LSTM with GWO
ITD-AATSU [14]	2021.	Short answer	Python, NLP, LSTM
SSSC-UTG/CMIE-GUCM [15]	2020.	Short answer	Python, Semantic Matching, CNN

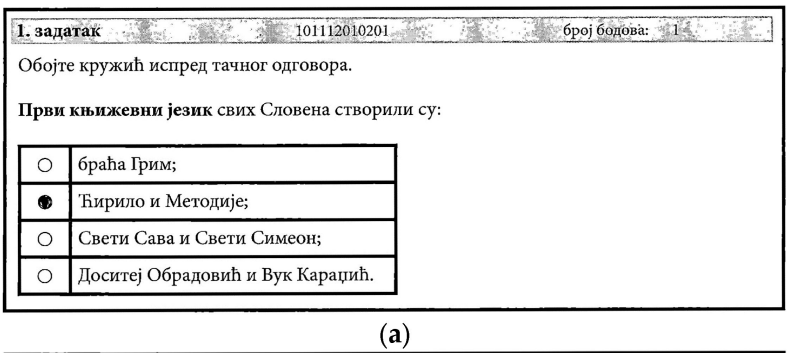
2.1. Multiple-Choice

Multiple-choice questions are widely employed in exams as they offer a versatile format. Such questions typically consist of a stem or partial statement, along with multiple answer choices. Another category within this class is True/False questions. The answer choices in a multiple-choice question include one or more correct options, as well as incorrect alternatives known as distractors. This streamlined approach allows candidates to swiftly answer such questions, making them particularly suitable for assessing a broad range of knowledge.

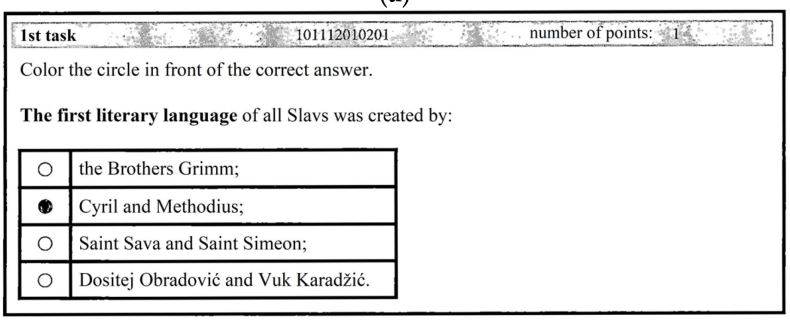
All the systems capable of automatically assessing multiple-choice question types employ diverse computer vision algorithms during the processing phase. Each system has indicated the technologies they utilize and are implemented in programming languages such as Python or MATLAB.

The majority of selected systems utilize the OpenCV library in conjunction with proprietary algorithms to detect different shapes (such as circles and rectangles) that represent question-and-answer regions. Some systems (DTE-UCM, for instance) perform the recognition process on a template test utilizing the Hough transform [16], translating the identified question and answer structure onto filled-in tests by aligning the template with the test boundaries. Conversely, other systems (SEE-UB, for instance) solely focus on detecting regions of interest on filled-in tests.

Regarding accuracy in grading, all selected systems demonstrate astonishingly high levels of precision: SEE-UB (99.9%), DTE-UCM (99.4%), and SE-ECU (95–100%). Among them, the SEE-UB system stands out as the fastest, processing a test sheet in approximately 250 milliseconds, while the slowest system requires several seconds. It should be noted, however, that the number of questions and answers on the answer sheets varies, and some systems impose limitations on the quantity allowed per test sheet. Additionally, all systems, except for the SEE-UB system, require questions and answers to be separated on distinct sheets of paper. In our evaluation, the system developed by the authors (the SEE-UB) has demonstrated the most impressive results. An example of a multiple-choice question can be seen in Figure 1a (Figure 1b is the English-translated version of Figure 1a).



(**a**)



(**b**)

Figure 1. (**a**) Multiple-choice question in Serbian. (**b**) Multiple-choice questions translated into English.

2.2. Matching

Matching questions are commonly employed in exams as an effective method for evaluating relationships between concepts. Typically, these questions consist of two sides, with statements or stems occupying the left side, and corresponding answers or choices on the right side. Moreover, this question type can be presented in a textual format where it becomes imperative to underline the words that correspond to the answer to the provided question.

Matching questions are typically employed when detailed knowledge assessment is crucial. Additionally, multiple-choice questions that provide designated answer placeholders, requiring students to write the selected answer letter or number, are also classified within this question category. This decision stems from the fact that by increasing the number of answer placeholders in these questions, they can be easily generalized to the Matching question format.

The number of systems capable of grading matching questions is relatively limited. The SEIE-FU system is implemented using the Python programming language, while the PMFU system additionally utilizes MATLAB programming language besides other tools. Each of the chosen systems utilizes Python libraries for convolutional neural networks, accompanied by proprietary algorithms to detect regions containing questions and answers. Both systems perform recognition exclusively on filled-in tests. The SEIE-FU system employs the YOLO [17] convolutional network, while the PMFU system develops its own convolutional neural network.

The PMFU system demonstrates a high grading accuracy of at least 92%. The SEIE-FU system did not explicitly state the achieved accuracy in grading. None of the selected systems provided information regarding the processing time required for a single test sheet. Additionally, none of the systems impose limitations on the number of questions and answers per test sheet.

In our assessment, the SEIE-FU system has showcased superior results. It effectively handles issues related to incorrect recognition caused by scribbles, and it does not restrict students in terms of where they can provide their answers. Furthermore, the system supports multiple languages in addition to English. On the other hand, the PMFU system encounters challenges when processing slightly tilted scanned test images, leading to the exclusion of those particular tests. An example of a matching question can be seen in Figure 2a (Figure 2b is the English-translated version of Figure 2a).

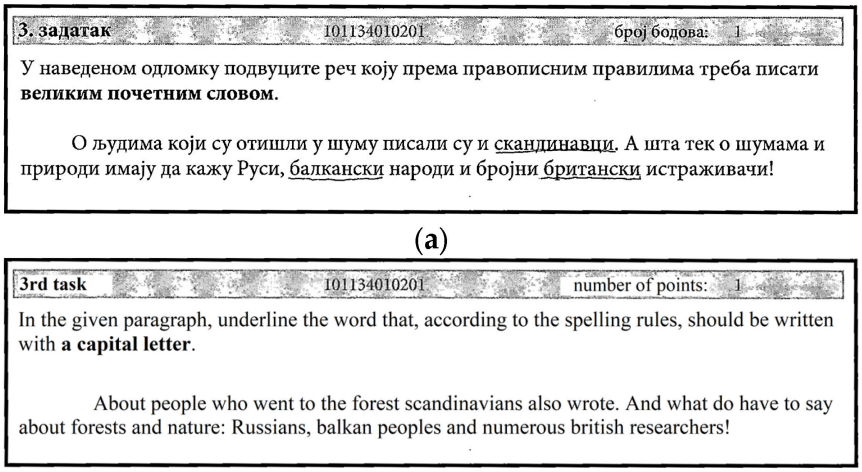


Figure 2. (**a**) Matching question in Serbian. (**b**) Matching questions translated into English.

2.3. Short Answer

Short answer questions, also known as fill-in or completion questions, serve as a valuable tool for evaluating candidates' comprehension of fundamental principles. Typically,

these questions consist of a concise statement or paragraph with blank placeholders that must be filled in. These types of questions are usually employed when assessing candidates' depth of knowledge.

All of the systems included in the selection employ some form of artificial intelligence, utilizing a range of techniques such as machine learning, natural language processing, deep learning, relation networks, long short-term memory, bidirectional encoder representations from transformers, semantic matching, and text mining. Among the selected systems, FCAI-BU and SSSC-UTG/CMIE-GUCM utilize convolutional neural networks to detect handwritten short answers on paper tests. In contrast, other systems perform the grading process directly on the extracted text from digital images or input forms, without involving the recognition of handwritten text from paper test images.

Regarding the reported grading accuracy, not all systems disclose the achieved values. FCAI-BU presents a Pearson score ranging from 0.77 to 0.95, ITD-AATSU reports a score of 0.95, and SSSC-UTG/CMIE-GUCM demonstrates 95% accuracy in grading. None of the systems provide information on the processing time required for grading tests.

In our assessment, the SSSC-UTG/CMIE-GUCM system has displayed the most favorable outcomes. It showcases remarkable accuracy in grading and incorporates a handwriting text recognition module, supported by a preprocessing module that assists in locating the handwritten answers and performs various image manipulation operations to enhance the accuracy of text recognition. An example of a short answer question can be seen in Figure 3a (Figure 3b is the English-translated version of Figure 3a).

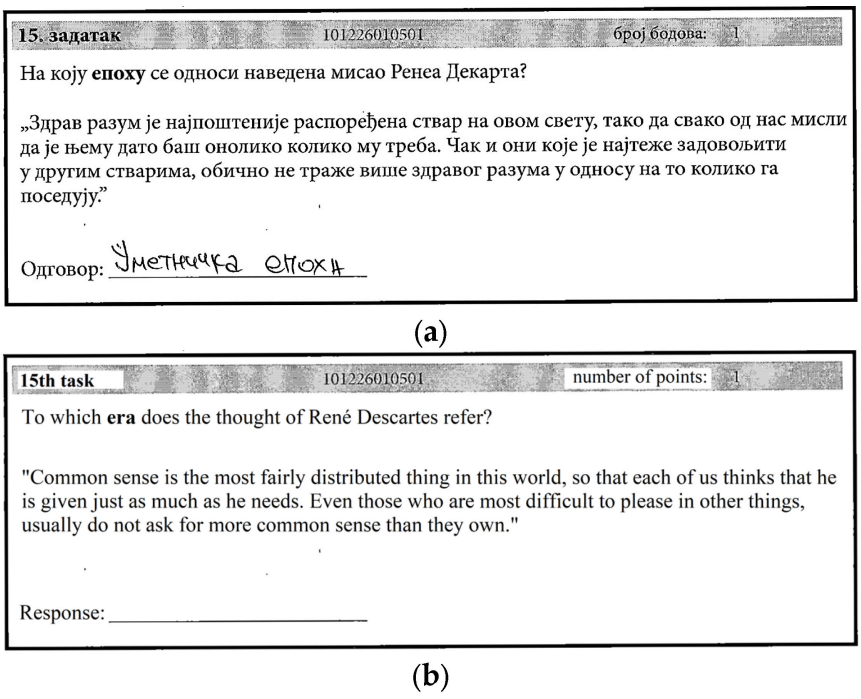


Figure 3. (**a**) Short answer question in Serbian. (**b**) Short answer questions translated into English.

2.4. Related Work Analysis and Solution Proposal

After an extensive examination of the relevant literature, it is evident that software systems designed for the automated grading of paper-based tests are often specialized in assessing only one question class. This specialization contributes to their impressive accuracy in grading. Notably, these systems are primarily configured to evaluate tests comprising a single-question class.

However, when crafting assessments, the goal is to introduce a variety of question classes to assess candidates' understanding from diverse angles. Each question class engages candidates uniquely when they decide on their chosen responses. Moreover,

since the limitations of one question type can be balanced by the strengths of another, the significance of test diversity, encompassing various question classes, becomes paramount.

Therefore, it becomes highly imperative to establish a system capable of automatically evaluating as many question classes within a test as feasible, thereby accommodating a broader spectrum of tests. Furthermore, it is crucial for this system to facilitate the composition or selection of answers directly within the designated answer space for each question, rather than relying on separate forms for response submission. This approach enhances candidates' focus on the question content and their responses while substantially reducing the likelihood of errors compared to tests featuring separate forms for questions, answers, and response submission.

3. Software System's Architecture

Automated paper test-scoring software systems are typically designed to evaluate specific question types. However, a consistent pattern has been observed in the sequence of stages involved in the process, beginning with test generation and concluding with result acquisition. The variations lie in the specific implementation details of each stage, particularly the phase associated with question processing.

The process commences by creating a digital version of the test, adhering to the specified constraints outlined in the configuration file. Subsequently, the digital test is replicated into multiple paper copies through copy centers. Blank paper tests are then dispatched to the respective institutions responsible for administering the candidate examinations. Upon completion of the tests, candidates fill out the paper-based answer sheets. Subsequently, the filled-in paper tests are collected and transported to scanning centers where machines digitize the filled paper tests, generating digital representations of the tests. These digital test forms serve as the input for the primary processing phase of the system, responsible for identifying questions and their corresponding answers, as well as processing them into a suitable format for later entry into the candidate database, where individual results are stored. It is feasible to establish a generalized process flow diagram encompassing the essential stages of the system, which is illustrated in Figure 4.

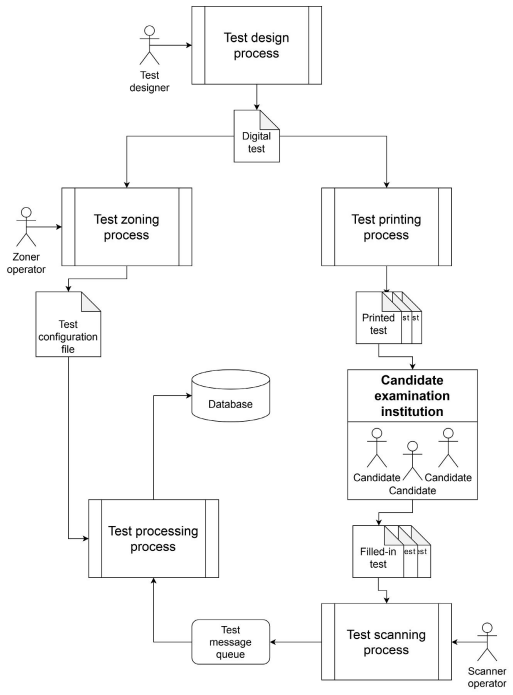


Figure 4. Generalized flow diagram.

The entire system comprises three fundamental components: zoning, scanning, and processing.

The zoning component is realized as an application that takes a digital blank test as input. A zoner operator is responsible for identifying and marking the QR code and barcode zones on the initial page, which contain the test and candidate information, respectively. Additionally, the upper left border region is designated as the reference point for all subsequent regions of interest, encompassing the questions and their corresponding answers. On subsequent pages, QR codes are present, carrying information about the data on each page. Subsequently, the zoner operator selects and outlines rectangular zones for the questions and their respective answers by utilizing mouse-dragging techniques on each page. The operator also specifies the question type and verifies the captured data.

Zoning is performed on a blank test devoid of question-and-answer text, primarily for security reasons. The zoning process generates a zoned test configuration file, which stores crucial details such as the number of pages, the quantity and types of questions on each page, and the approximate locations of the questions and their answers. This information is solely utilized for validation purposes within the processing component, which is responsible for locating the regions of interest.

The scanning component is realized as an application designed to handle filled-in paper tests. It operates by accepting the filled-in test sheets as input. The scanning component of the system takes over, performing QR code recognition and arranging the test pages in the correct order. The scanned test pages are then assembled into a multi-page .tiff file.

Each scanned test can fall into one of three categories: complete, incomplete, or erroneous. Fully successful scans, representing complete tests, are sent to a dedicated repository on the server that stores the complete tests. Incomplete tests, which are missing one or more pages, are also sent to a specific repository for incomplete tests. While they are forwarded for further processing, manual intervention becomes necessary to address the missing pages. Tests with errors occur when the QR codes or barcodes cannot be read or when the test pages exhibit significant deformities. Such tests are not dispatched for processing and require manual intervention.

For each scanned test, the scanner component sends messages to the associated message queue. Each message contains pertinent information and, additionally, the component periodically transmits statistical information regarding the number of complete, incomplete, and erroneous tests. Furthermore, when scanning the same test multiple times, distinct messages are dispatched to the corresponding message queue, resulting in the processing component handling the same test on multiple occasions.

The test processing component of the system has been designed as a versatile service capable of running in multiple instances. Each instance of the service establishes a pool of processing threads, with each thread being assigned to an incoming message queue. For every scanned and completed test, a corresponding message is inserted into the incoming message queue by the scanning component. These messages contain the file path to the digitally scanned test stored on the server, as well as the path to the configuration file specific to that test.

Upon successful processing of a test, the resulting data and outcomes are stored in the directory dedicated to completed tests, residing within the corresponding server directory for that particular test. Upon receipt of a message, each processing thread meticulously deciphers its contents, subsequently retrieving the designated test and configuration file from the specified path. Once these essential components are obtained, the intricate process of test and question processing commences, whereby the appropriate procedures are employed based on the specific question types encountered.

4. Processing Component Implementation Details

Initially, the system first identifies the barcode sticker, which serves to uniquely identify the candidate, affixed to the cover page by the human exam proctor. Subsequently, each

individual page is scanned for a QR code. These QR codes are inserted during the test design phase and contain information pertaining to the test identification and the respective page number. Since these QR codes are incorporated into every page during test design, they are inherently well-oriented. This orientation allows for the system to accurately determine any necessary rotation adjustments needed to correct any inadvertent page rotation that may occur during the digitization process of the test.

Prior to initiating the question processing phase, it is imperative to identify the specific regions encompassing the questions and their corresponding answers. These regions are characterized by rectangular shapes and are positioned between four borders placed at each corner of the test page, forming a large letter L shape. Each test is represented by a multi-page .tiff file, where each image within the file corresponds to a single page of the test. To enhance the recognition process, several morphological operations need to be applied to the images.

At first, the test image undergoes grayscale conversion, as the subsequent algorithms necessitate this image format. Subsequently, a Gaussian blur is applied to the image using an appropriately sized rectangular kernel, effectively eliminating Gaussian noise. The image is then transformed into a binary inverted representation using an appropriate threshold function, employing the OTSU algorithm for the automated selection of the optimal threshold value. The inversion image is essential since the frames outlining the regions of interest in the original test are black, whereas shape detection algorithms require the shapes to be white. Morphological operations are then implemented to retain only horizontal and vertical lines, preserving rectangular regions encompassing the questions, answers, and borders within the image. Additionally, a closing morphological operation is performed to address any discontinuities in shapes of interest introduced during the scanning process. With these transformations completed, the search for regions of interest can be initiated. A search for shapes within the image yields a list of contours, which are defined as lines that enclose regions consisting of pixels with the same intensity. In the context of a binary image, where pixels can have either a value of 1 or 0, contours outline areas where the intensity remains constant (with a value of 1 in the case of a binary inverted image).

The identification of answer regions varies depending on the question type. Furthermore, the approaches to processing answers also exhibit variations. The subsequent subsections will elaborate on the techniques employed for their detection. Algorithm 1 illustrates the pseudocode for the processing component.

Algorithm 1. Pseudocode for the processing component

```
try:
  test = load_test_tiff()
  config = load_config()
  result = create_result_file()
  for page_num, page in enumerate(test):
    try:
      if not page_num:
        bc = get_bar_code(config, page)
        update_result(result, bc)
        if not bc:
          raise BarcodeNotFound()
        continue
      qr = get_qr_code(config, page, page_num + 1)
      update_result(result, qr)
      if not qr:
        raise QRcodeNotFound()
      page = rotate_image(page, qr.get_rotation_angle())
      morphed_page = morph_image(page)
      borders = get_borders(morphed_page)
      questions = get_questions(config, page_num + 1)
      potential_questions = get_potential_questions(morphed_page, borders)
      adjust_questions_locations(config, borders,
                                 questions, potential_questions)
      for q in questions:
        answers = get_answers(morphed_page, q)
        for a in answers:
          if q.type() is QUESTIONS.MULTIPLE_CHOICE:
            answer_results = get_circles(page, a)
          elif q.type() is QUESTIONS.MATCHING:
            answer_results = get_matches(page, a)
          elif q.type() is QUESTIONS.SHORT_ANSWER:
            answer_results = get_short_written(page, a)
          update_result(result, a, answer_results)
          save_answer_image(page, a)
        update_result(result, q)
        save_question_image(page, q)
    except Exception as e:
      log_error(f'Page: {e}')
      update_result(result)
      save_image_page(page)
  save_result(result)
except Exception as e:
  log_error(f'Test: {e}')
```

4.1. Multiple-Choice

The system is capable of processing questions that require selecting one or more provided answers by blackening the corresponding circles. These types of questions involve zoning the answer region, which is achieved through the zoning process using the zoner operator. Algorithm 2 provides the pseudocode for the processing component responsible for managing multiple-choice questions, while Figure 5 visually details the sequential steps involved in handling this question type on a test page, connected with arrows to illustrate the flow.

Algorithm 2. Processing component pseudo code for handling multiple-choice questions

```
def get_circles(page, answer):
    contours = find_contours(page)
    table_cells = get_answer_table_cells(answer, contours)
    if not check_table_grid(table_cells, answer):
     raise TableGrid()
    circles = []
    for cell in table_cells:
      for c in contours:
        if is_parent(cell, c)
        and is_circle(c, 0.25, 0.05)
        and is_centered(c, cell, 0.4, 0.6):
          circles.append(minimum_enclosing_circle(c))
          contours.remove(c)
          break
    if len(circles) < thresh:
      raise NotEnoughCircles()
    tresh = get_thresh(answer)
    circles = sorted(circles, key=lambda c: c.surface(), reverse=True)[:tresh]
    mean_radius = mean([c.radius() for c in circles])
    min_radius, max_radius = 0.9 * mean_radius, 1.5 * mean_radius
    circles = [c for c in circles if is_between(c, min_radius, max_radius)]
    if len(circles) < thresh:
      raise NotEnoughCircles()
    if not check_circles_grid(circles, answer):
      raise CirclesGrid()
    filled = []
    low, high = 1, 0
    for c in circles:
      ratio = 1. * count_black_pixels(page, c) / (c.get_radius() ** 2 * math.PI)
      filled.append(ratio)
      low, high = min(low, ratio), max(high, ratio)
    lower = median(0, 0.4, low + 0.4 * (high - low))
    upper = median(0.2, 0.6, low + 0.6 * (high - low))
    status = []
    for c, f in zip(circles, filled):
      status.append((c, None
                        if lower < f < upper
                        else
                        True if f > upper else False, f)
                    )
    return get_grid(status, answer)
except  Exception as e:
    log_error(e)
    return None
```

The answer regions in multiple-choice questions are situated within the question regions and are identified through the use of tables. The process of determining these answer tables, which are comprised of rectangular cells, involves conducting a search for rectangular contours within the question region. In addition to the location of each contour, the tree search method yields information about the hierarchical parent-child-sibling relations between the contours. Subsequently, a list of rectangular contours is obtained, which can potentially form a rectangular answer table.

Upon confirming the answer tables for a given question, the next step involves identifying the answer circles that need to be marked in order to answer the question. Each answer circle is represented by a single contour located at the center of a table cell. While one could simply check for darkened central regions within each cell, an additional search is conducted to detect the contours of the response circles. This approach ensures that the marking of answers by students is not specific and verifies adherence to the specified method of filling, as outlined in the test instructions.

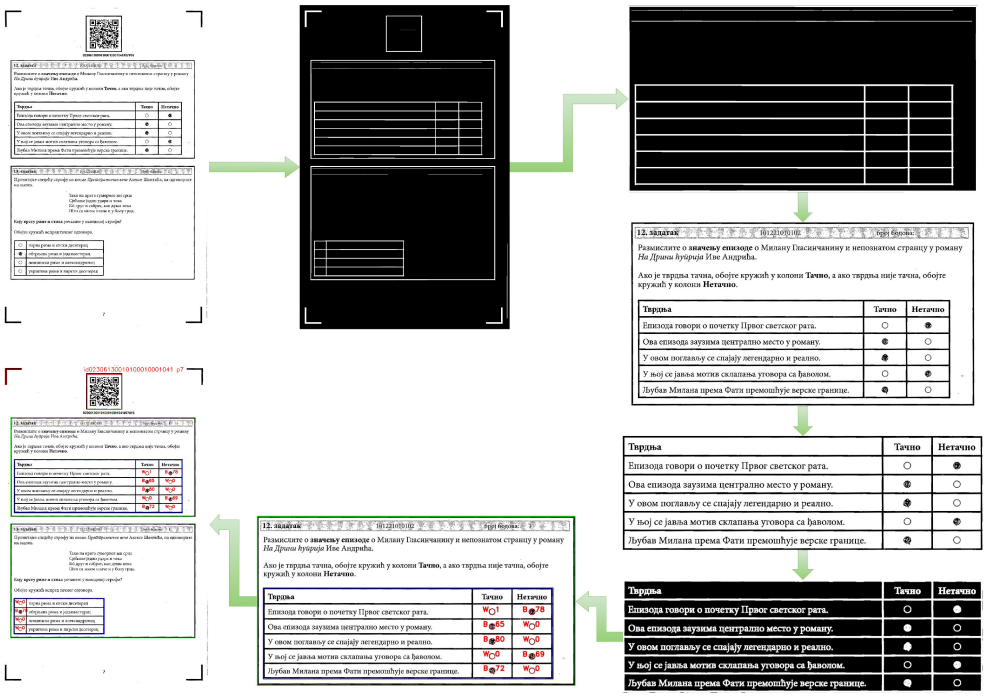


Figure 5. The sequence of processing multiple-choice question type on a test page.

The process of circle detection involves various mathematical operations and checks. For each potential circle contour, the perimeter is calculated, and based on this value, a reliability index is determined. This index is computed as the ratio of 4 times PI times the area of the contour to the square of its perimeter. A perfect circle would yield an index value of 1. To further validate the contour, the moment of the candidate circle is calculated, aiding in determining the contour's center of mass. The distance from the center of mass to each point on the contour is then calculated, and the median value is obtained. It is assessed whether a sufficient number of contour edge points meet the condition that their distance to the center of mass falls within a predetermined epsilon neighborhood of the previously computed median value. The selection of thresholds is based on experimental determinations. Algorithm 3 presents the process for testing and verifying circles.

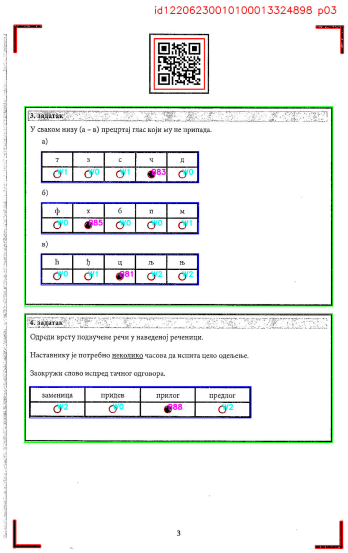
Algorithm 3. Pseudocode within the processing component designed for evaluating circle candidates

```
def is_circle(candidate, thresh, eps):
    num = 4 * math.PI * get_contour_area(candidate)
    den = candidate.get_perimiter() ** 2
    if not ((1 - thresh) <= num / den <= (1 + thresh)):
        return False
    center, points = candidate.get_moments(), candidate.get_points()
    median = median([get_distance(center, p) for p in points])
    dists = [
             abs(distance(center, p) - median_dist) / median_dist < eps
             for p in c.get_points()
            ]
    if sum(dists) <= len(dists) * thresh:
        return False
    return True
```

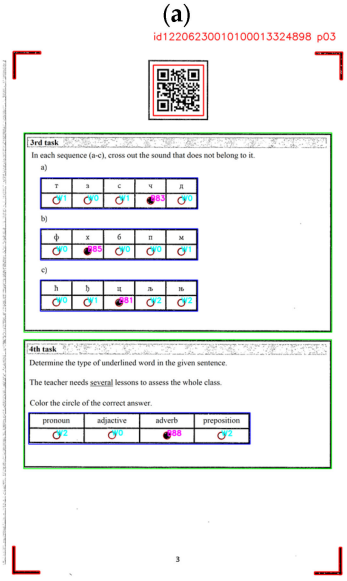
If the number of detected circles is insufficient for the given question, an error is recorded, indicating that an insufficient number of circles were detected. Conversely, if the number of circles exceeds the requirement for the question, the N largest circles (where N represents the number of answer circles expected for that question) are chosen based on their area. This selection accounts for situations where certain letters (e.g., "o") may be

recognized as circles. However, there is a constraint that the answer circles must be larger than any other letters forming the answer word within the table. Finally, a final check is conducted to ensure that the identified circles adhere to the criteria of forming a regular grid.

Once the grid of answer circles for a question is established, the process of determining the fill status of each circle begins. Initially, a portion of the square image containing the contour of each circle is extracted. Subsequently, the number of black pixels within the image, which correspond to the area inside the circle, is computed. However, it is important to note that even unfilled circles exhibit a small percentage of black pixels due to the contribution of the circle's borders. Additionally, students may vary in their degree of shading when filling the circles. To address these factors, a min-max scaling approach is employed to derive a score for the level of circle filling. It is important to consider that scenarios may arise where either all circles are filled, or none are filled, and they are handled appropriately. Figure 6a displays the multiple-choice question featured in the test (Figure 6b is the English-translated version of Figure 6a).



(**a**)



(**b**)

Figure 6. (**a**) Example of a multiple-choice question in Serbian. (**b**) Example of a multiple-choice question translated into English.

4.2. Matching

For questions of this type in the reference test, the format involves underlining one or more correct words within the free text of the answer. The answer region of the question corresponds to the question region itself and is not zoned by the zoner operator during the zoning process. Algorithm 4 illustrates the pseudocode for the processing component responsible for managing matching questions, while Figure 7 visually details the sequential steps involved in handling this question type on a test page, connected with arrows to illustrate the flow.

Algorithm 4. Processing component pseudo code for handling matching questions

```
def get_word_status(under_word_image, word, thresh, mode):
    image_segments = segment_image(under_word_image, word)
    segments = []
    for seg in image_segments:
        segments.append(count_black_pixels(seg))
    return is_underlined(segments, word.get_segments(mode), 0.5, thresh)

def get_matches(page, answer, thresh1=3., thresh2=0.65):
    template_answer_image, template_words = get_or_load_template(answer)
    answer_image = get_answer_image(page, answer)
    aligned_answer_image = align_image(answer_image, template_answer_image)
    matches = {}
    for t_word in template_words:
        morphed_image = morph_image(aligned_answer_image)
        under_word_image = get_under_region(morphed_image, t_word)
        flag = get_word_status(under_word_image, t_word, thresh1, 'under')
        if not flag:
            lines_image = morph_image_horizontal_lines(aligned_answer_image)
            under_word_image = get_adjusted_under_region(lines_image, t_word)
            flag = get_word_status(under_word_image, t_word, thresh2, 'under_line')
        matches[t_word] = flag
    return matches
```

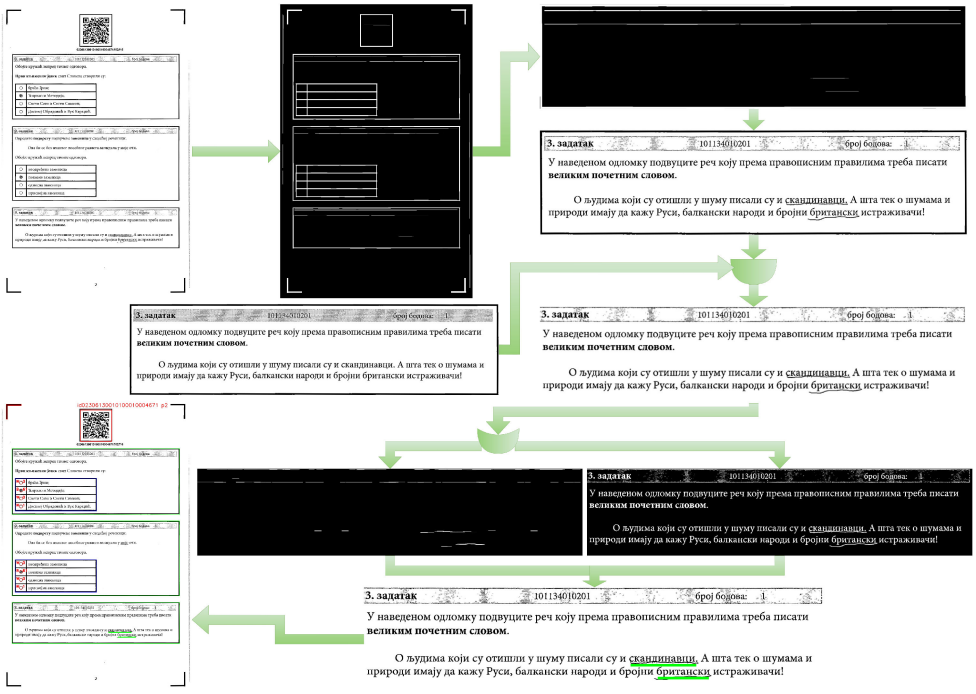


Figure 7. The sequence of processing matching question type on a test page.

The initial step entails localizing and cropping the image of the question region of interest from the blank template test. This image is then converted to grayscale and subsequently

binarized to facilitate further processing. The Tesseract optical character recognition (OCR) engine is employed to extract the words from the question's and answer's text, determining their positions and dimensions. The words are organized into paragraphs and within each paragraph, they are arranged into lines according to their appearance order. This segmentation aids in identifying the regions below the text lines where pencil underlining may occur.

Two distinct methods are utilized to determine whether a word is underlined, with differences in the underlining regions. In the first method, the underlining regions span from the baseline of one text line to the top line of the next text line within the same paragraph. If the word is the last line of the paragraph, the underlining extends to the beginning of the subsequent paragraph or the end of the question region if it is the last line of the last paragraph. In the second method, the top edge of the underlining region is adjusted to align with the middle of the text line. In both cases, the underlining region of each word is divided into an appropriate odd number of segments, and the number of blackened pixels is calculated. A distinction between the two methods is that in the first method, the number of blackened pixels is computed on the binary question image, while in the second method, the question image undergoes morphological transformation to retain only horizontal lines. This transformation is achieved by applying an appropriate kernel to preserve lines of at least one letter width.

Both methods are employed to determine if a word is underlined, with the second method utilized only when the first method indicates that the word is not underlined. The first method suffices because observing the difference in pixel darkness within the underlining region of the question image in the template blank test compared to the filled test has proven to be effective. The second method is necessary as candidates often underline the word in a way that the underlining line passes through the word rather than beneath it. Thus, the underlying region's location in the second method, as previously mentioned, is adjusted. Additionally, the first method yields inadequate results when the underlining line is obscured by black pixels forming the letters of the word, and the difference in pixel darkness between the template test and the specific filled-in test sample is insufficient.

To observe the difference in pixel darkness within the underlined region between the template and the filled-in test, it is necessary to search for the words in the filled-in test or transform the question image from the filled-in test after successfully identifying the question region. The second method is chosen due to its efficiency, as underlining the word impedes the accurate recognition of the word's position and dimensions. The transformation of the filled-in test question image involves converting it to grayscale, followed by binarization. The ORB (Oriented FAST and Rotated BRIEF) detector is then initialized to detect key points and descriptors in both the template test and the filled-in test images. Given the similarity between the images and their negligible rotation angles, a simple detector like ORB is sufficient. Subsequently, a BFMatcher (Brute-force matcher) object is created, specifying the Hamming distance as the distance criterion for the similarity between the two sets of descriptors. Using the matcher object and the descriptors, the locations where the matching occurs are determined, and the best matches (smallest distance) are selected. These matches are used to find a perspective transformation that aligns the filled-in test question image with the template test question image.

With the transformed question image in hand, the next step is to determine the level of pixel darkening within the underlined regions. The process is repeated using the two aforementioned methods, which were initially applied to the template test question image to establish a reference value for pixel darkness within the underlined region of each word. The obtained values from the filled-in test and the template test are compared. If the difference exceeds the reference value specified in the configuration file in at least half of the segments, the word is considered underlined. Notably, the blackness threshold is notably lower in the second method, taking into account that the morphological operation, which retains only horizontal lines on the test, eliminates a significant number of blackened

pixels. Figure 8 displays the matching question featured in the test (Figure 8 will not be translated into English as it depicts the identical question as presented in Figure 2a).

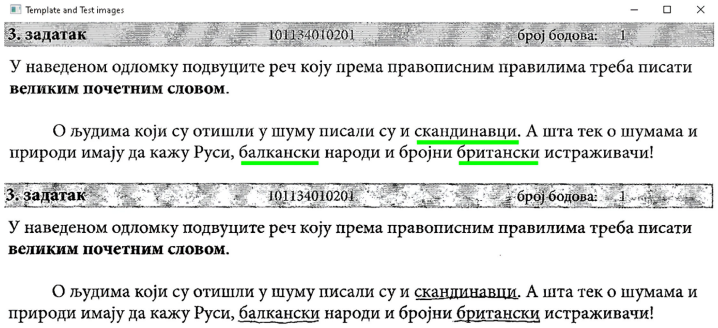


Figure 8. Example of a matching question.

4.3. Short Answer

The system has the capability to process short answer questions, which require writing a response on the designated line for the answer. The answer zone for this question type is determined through the zoning process using the zoner operator. Algorithm 5 presents the pseudocode for the processing component responsible for managing short answer questions, while Figure 9 visually details the sequential steps involved in handling this question type on a test page, connected with arrows to illustrate the flow.

Algorithm 5. Processing component pseudo code for handling short answer questions

```
def get_short_written(page, answer):
    answer_image = get_answer_image(page, answer)
    lines_image = morph_image_horizontal_lines(answer_image)
    lines = find_contours(lines_image)
    answer_lines = find_answer_lines(answer, lines)
    morphed_image = erase_lines(answer_image, answer_lines)
    morphed_image = repair_image(morphed_image)
    hw_model = get_or_load_model()
    result = []
    for line in answer_lines:
      seg_image = segment_image(morphed_image, line)
      contours = find_contours(seg_image)
      contours = sorted(contours, key=lambda c:  c.get_x(), reverse=False)
      digits = []
      for c in contours:
        c_image = segment_image(seg_image, c)
        d = hw_model.decode(c_image)
        digits.append(d)
      result.append((line, ''.join([str(d) for d in digits])))
    return result
```

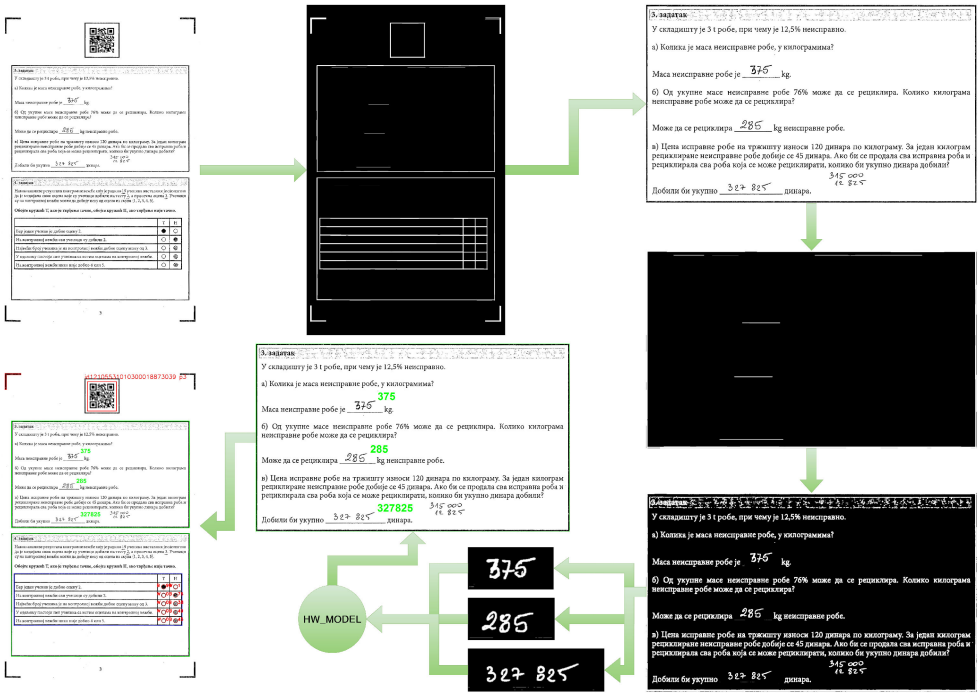


Figure 9. The sequence of processing short answer question type on a test page.

The first step involves identifying and extracting the image of the question region of interest from the filled-in test. This image is then converted to grayscale and subsequently transformed into a binary format to aid in subsequent processing. Then, a morphological operation is applied to the image of the question, resulting in only the horizontal lines remaining visible. This is achieved by applying an appropriate kernel element with dimensions specified in the corresponding configuration file for that test. Next, contours representing the horizontal lines are detected, and their location information is overlapped with the answer regions provided by the zoning process for each question. Once the horizontal lines for answer writing are determined, an image processing operation is performed to erase the answer-writing line from the image. Following that, a morphological closing operation is applied to potentially recover any extended portions of the written answer, considering that candidates often write their answers in a way that extends not only above the line (as it should) but also below it, effectively intersecting with the answer line. Then, the answer region is extracted from the image, and a search for digit contours is conducted. Each character is extracted from the image and transformed into a suitable format for the model used for the recognition of the characters. This process entails resizing the recognized region of the characters, aligning it to the center, and adding padding to the smaller dimension to achieve a final size of 32 pixels by 32 pixels, which matches the input data requirements of the network model.

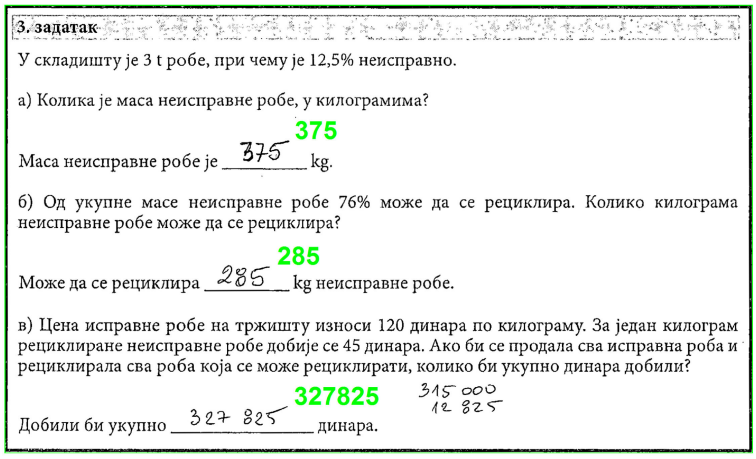
For the purpose of digit recognition, a modified version of the convolutional neural network LENET5 [18] is utilized [19]. The original neural network model exhibits issues of significant bias and variance, and suitable optimizations have been implemented to address these challenges effectively [20–22]. The modified neural network model consists of two convolutional layers followed by a max-pooling layer with ReLU activation. These are then followed by two more convolutional layers and another max-pooling layer with ReLU activation. Afterward, three fully connected layers are employed, culminating in 10 outputs that utilize the softmax function.

Batch normalization is conducted after every two convolutional layers and after each fully connected layer to stabilize the network and promote faster convergence of the learning algorithm. After each pooling layer and the final fully connected layer, regularization is performed by randomly dropping a quarter of the neurons from the network to enhance

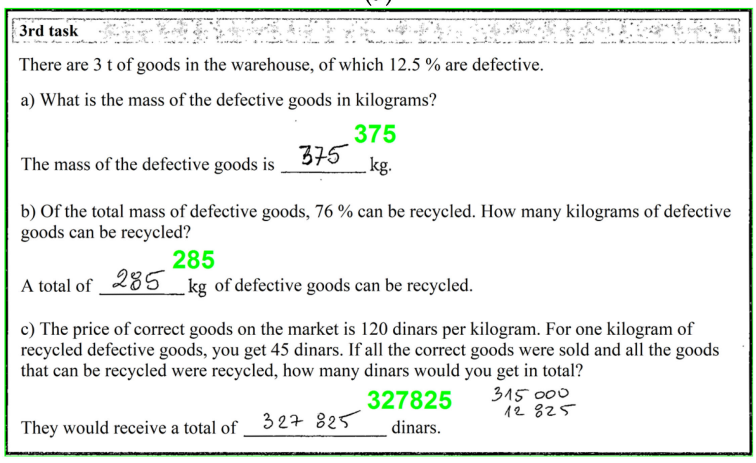
its generalization capability [23]. In the initial convolutional layers, L2 regularization is employed to penalize the error function more, aiming to reduce the complexity of the model and prevent overfitting. Along with data augmentation, all of the aforementioned measures are intended to reduce the model's variance [24,25].

The network is trained on data consisting of images from the MNIST dataset [26]. It comprises 60,000 training samples and 10,000 test samples. Data augmentation is applied by randomly rotating the images in both directions by no more than 10 percent, shifting them slightly in all four directions, and zooming in and out by a slight random percentage. Additionally, since the images represent pixel value matrices, standardization is performed by subtracting the mean and dividing by the standard deviation, which helps expedite the model training process.

Model bias reduction is achieved by adding more convolutional layers and increasing the number of filters in the convolutional layers. By creating a denser and deeper network with more hidden layers, a more complex model is formed that can better capture significant features in the input data. The model is trained over 25 epochs using a variable learning rate, which dynamically adjusts to facilitate rapid convergence of the learning algorithm when the model detects learning stagnation. Figure 10a displays the short answer question featured in the test (Figure 10b is the English-translated version of Figure 10a).



(**a**)



(**b**)

Figure 10. (**a**) Example of a short answer question in Serbian. (**b**) Example of a short answer question translated into English.

5. Evaluation and Discussion

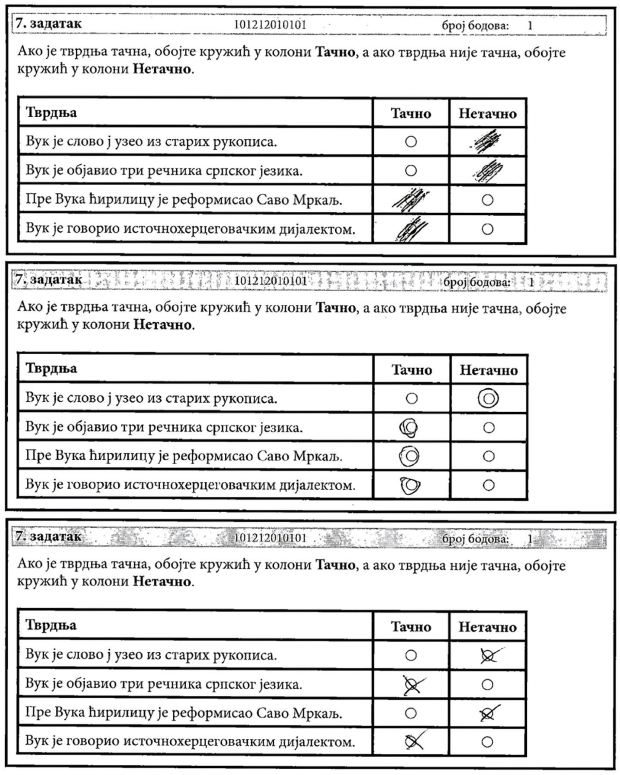
The system underwent testing and evaluation using a total of 43,225 tests in the native language. The test comprises a total of 16 pages, with the first page serving as the title

and identification page, while the final page is left blank. Within the test, the questions are distributed across these pages. The answer region is positioned within the question region. In certain cases, both the question and its corresponding answers may span multiple pages. In the case of multiple-choice questions, the number of answer options provided may vary. Similarly, for matching questions, the number of words to be underlined can range from a single word to multiple words. Each test consisted of 20 questions, with 17 being multiple-choice, 2 matching questions, and 1 short numerical answer question. Subsequently, the results for each question type will be presented.

5.1. Multiple-Choice

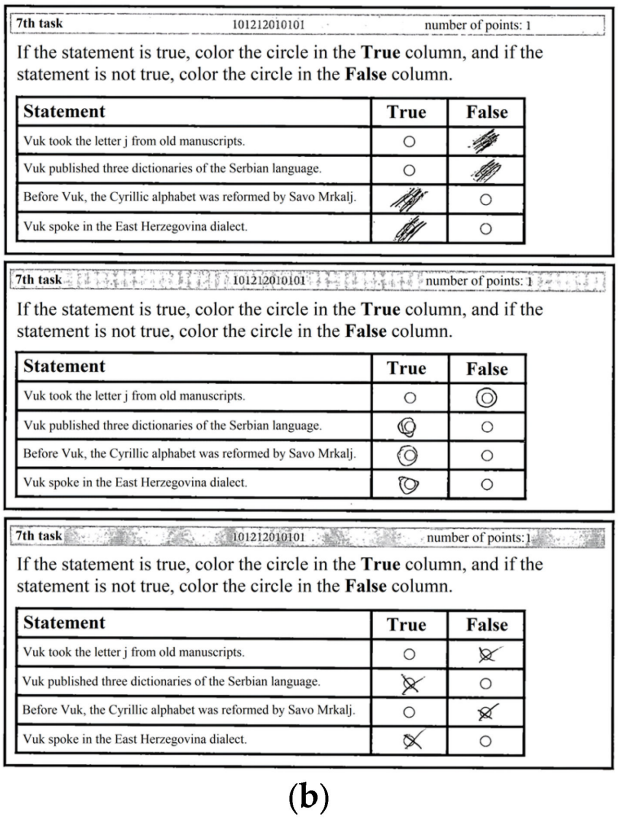
In the system testing process, a total of 734,825 multiple-choice questions were included. Among these questions, 734,487 question-and-answer regions were successfully detected, resulting in a 99.95% success rate in identifying question-and-answer regions. Furthermore, 734,132 grids of circles representing the answer choices to these questions were accurately recognized, representing a success rate of 99.91%.

However, in a small fraction of cases (0.09%), specifically 693 questions, the answer region was successfully identified, but the grid of circles was not correctly recognized and necessitated manual verification. Upon manual review, it was discovered that candidates often crossed out certain circles to invalidate previous answers, which caused the circles to go unrecognized. Additionally, some candidates encircled the answer choices, created black squares around the circles, or made additional markings on the circles, resulting in the omission of certain circles from recognition. Consequently, such questions were flagged for manual inspection to ensure accuracy. Figure 11a illustrates an instance where a particular response necessitates manual review (Figure 11b is the English-translated version of Figure 11a).



(**a**)

Figure 11. *Cont.*



(**b**)

Figure 11. (**a**) Responses needing a manual review in Serbian. (**b**) Responses needing a manual review translated into English.

It was anticipated that the cumulative count of answer circles would reach 3,319,086, incorporating 734,487 recognized answer regions. After conducting a thorough examination, the actual number of identified answer circles was 3,316,215, showcasing an exceptional precision rate of 99.91% in the recognition of answer circles.

Out of a total of 734,132 successfully recognized grids of circles, 733,987 grids accurately had all of their circles' fullness correctly identified, resulting in a remarkable rate of 99.98%. Considering the 3,316,926 circles present in these 733,987 grids, an outstanding 99.99% of the circles were correctly identified with their fullness status, totaling 3,316,458 circles.

5.2. Matching

During the system testing phase, the number of questions classified as "matching type" was 86,450. Out of this total, 86,391 question-and-answer regions were successfully identified, resulting in a success rate of 99.93% in detecting question-and-answer regions. Moreover, 86,373 answers to these questions were accurately recognized, representing a success rate of 99.98%.

However, for a small subset of 28 questions (0.02%), although the question regions were successfully detected, not only the underlined words within those questions were recognized, or not all of the underlined words were detected. A manual review of these tests revealed that candidates often crossed out one of the words to invalidate a previous answer, which inadvertently led to additional words being recognized. Certain candidates utilized excessively thin and barely discernible underlines on words, resulting in their non-recognition. Figure 12 illustrates one such instance (Figure 12 will not be translated into English as it depicts the identical question as presented in Figure 2a).

In the first question, one word should have been underlined, while in the second question, seven words should have been underlined. Nevertheless, it is important to note that not all candidates provided correct answers or completed the question accurately. As a result, the total number of underlined words reached a quantity of 781,779. The

recognition process successfully identified 781,721 words, achieving a recognition rate of 99.99%. Upon closer examination, it was found that there were 29 redundantly recognized and 58 unrecognized words distributed among the 28 questions.

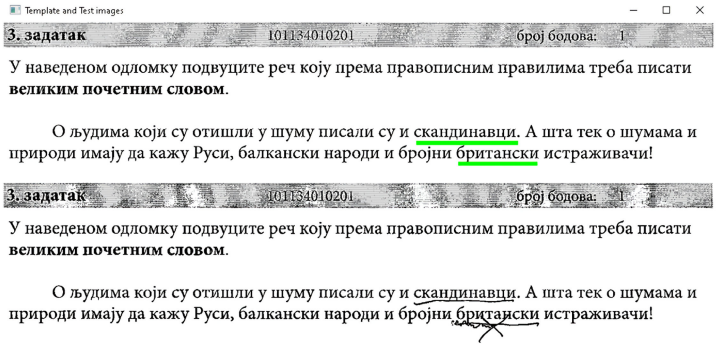


Figure 12. Unsuccessful underlined word detection.

5.3. Short Answer

In the system testing process, there were a total of 43,225 short numeric answer type questions, each containing 3 answer regions, resulting in a total of 129,675 regions. Among these questions, 129,599 question-and-answer regions were successfully detected, resulting in a success rate of 99.94% in detecting these regions. Out of these, 123,715 answers (95.46%) were successfully recognized. However, for 5884 answers (4.54%), although the region of the question and answer was successfully recognized, not all digits of the given answer were accurately recognized.

This particular question was designed with three designated areas for inputting numerical answers, with the expectation that the correct answer would consist of a total of twelve digits. However, some candidates did not answer the question correctly or provide the correct number of digits. Additionally, some candidates did not attempt to solve the question at all. As a result, the total number of recognized digits amounted to 494,460. Out of these digits, 488,181 were successfully recognized, achieving a recognition rate of 98.73%. The achieved recognition rate, 98.73%, is slightly lower compared to the percentages obtained on the MNIST dataset for both the training and test data sets. On the MNIST dataset, the recognition rates were 99.58% and 99.37%, respectively.

5.4. Summary

Table 2 provides a comprehensive summary of the performance data acquired during the evaluation process of the software system, categorized according to each question type that the system is capable of assessing.

Table 2. A comprehensive summary of the evaluation process.

Question Class	Total Question and Answer Regions	Correctly Identified Question and Answer Regions	%	Correctly Recognized Answers	% (% Total)	Total Number of Elements to be Recognized (Circles, Words, Digits)	Correctly Recognized Elements (Circles, Words, Digits)	%
Multiple-choice	734,825	734,132	99.91	733,987	99.98 (99.89)	3,316,926	3,316,458	99.99
Matching	86,450	86,391	99.93	86,373	99.98 (99.91)	781,779	781,721	99.99
Short answer	129,675	129,599	99.94	123,715	95.46 (95,40)	494,460	488,181	98.73

In the table provided, for answer recognition in each question type, two values are presented: "Correctly recognized answers" and "Correctly recognized elements (circles, words, digits)". These values vary for each question type, with the second value being higher. The disparity arises from the fact that each question entails a larger number of

elements comprising the answer. Consequently, the recognition of an individual element, relative to the entirety of elements constituting a single answer, holds greater significance.

Upon examining the existing software systems discussed in Section 2, it becomes evident that none of them possesses the capability to comprehensively assess paper-based tests encompassing various question types, including multiple-choice, matching, and short-answer questions. Additionally, the majority of the previously mentioned systems are not well suited for the general paper test format; notably, they are engineered to handle tests with separate answer sheets. Moreover, adapting them for the assessment of standard paper test formats necessitates substantial code modification with questionable performance. The system we have described and implemented stands as the sole solution offering this unique capability. Furthermore, employing the methodology we have outlined for processing the entire test bestows an additional advantage when compared to the systems that were evaluated.

Furthermore, this enhanced precision can be attributed to the utilization of sophisticated algorithms. These algorithms are employed for assessing the correct geometric shape, with the added benefit of an adaptive threshold for determining the completeness of the desired shape, resulting in notably accurate outcomes for multiple-choice questions. Additionally, the system benefits from algorithms that undergo morphological image adjustments and employ a template-answer image-matching approach, coupled with an additional check of candidates' answers, all of which collectively contribute to the system's accuracy in matching class-based questions. Furthermore, the system attains remarkable accuracy in assessing short-answer class questions through the implementation of modified algorithms designed for the detection of answer lines, the delineation of handwritten characters, and the integration of a specialized neural network for text prediction.

6. Conclusions

The primary objective of this study is to showcase the software system, which has demonstrated high precision in evaluating different question types and automatically recognizing marked answers on a combined question-and-answer paper format. The system introduces the capacity to evaluate multiple choice questions, and matching-type questions, wherein the answers require the candidate to underline specific words from the test, and it can evaluate short answer questions. To address these challenges, the system employs additional artificial intelligence techniques, including convolutional neural networks and computer vision. Moreover, it encompasses the capability to identify and categorize various types of errors, assigning them to different severity levels, and managing their processing accordingly.

However, it would be highly desirable to enhance the system's resilience and adaptability when confronted with challenges posed by suboptimal scan quality. By fortifying the system's capabilities in this regard, it would ensure consistent and reliable performance, even in less-than-ideal scanning conditions. Furthermore, the most significant challenge lies in enabling the system to grade essay-type questions. To tackle this issue, further augmentation of the system is needed, utilizing additional artificial intelligence techniques such as natural language processing [27]. It would also be advantageous for the system to verify the candidate's identity based on their handwriting [28,29]. These aspects present avenues for future research and development.

The system that has been put into operation is utilized during the assessment of students, although its applicability extends beyond academic settings. For instance, this system could find utility in assessing knowledge for purposes such as acquiring a driver's license. Furthermore, it can serve various purposes, including conducting large-scale paper-based surveys or collecting information on medical histories through medical questionnaires and other paper-based forms.

Author Contributions: Conceptualization, V.J.; Data curation, N.B.; Formal analysis, B.N.; Funding acquisition, B.N.; Investigation, V.J.; Methodology, B.N.; Project administration, B.N.; Resources, N.B.; Software, V.J.; Supervision, B.N.; Validation, N.B. and B.N.; Visualization, V.J.; Writing—original draft

preparation, V.J.; Writing—review and editing, B.N. and N.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported and funded by the Science Fund of the Republic of Serbia, grant no. 6526093, AI-AVANTES (www.fondzanauku.gov.rs). The authors gratefully acknowledge the support.

Data Availability Statement: The data supporting this study's findings belong to the Ministry of education of the Republic of Serbia. Due to this reason, the data cannot be public. The authors receive data under special terms for research purposes only. If needed, the authors can send a request to the Ministry to make data available to the editor to verify the submitted manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Jocović, V.; Đukić, J.; Mišić, M. First Experiences with Moodle and Coderunner Platforms in Programming Course. In Proceedings of the Tenth International Conference on e-Learning, Belgrade Metropolitan University, Belgrade, Serbia, 29 September 2019; pp. 81–86, ISBN 978-86-89755-18-3.
2. Lewis, I.; Watson, B.; White, K.M. Internet versus paper-and-pencil survey methods in psychological experiments: Equivalence testing of participant responses to health-related messages. *Aust. J. Psychol.* **2009**, *61*, 107–116. [CrossRef]
3. Hüseyin, Ö.Z.; Özutran, T. Computer-based and paper-based testing: Does the test administration mode influence the reliability and validity of achievement tests? *J. Lang. Linguist. Stud.* **2018**, *14*, 67–85.
4. McClelland, T.; Cuevas, J. A comparison of computer based testing and paper and pencil testing in mathematics assessment. *Online J. New Horiz. Educ.* **2020**, *10*, 78–89.
5. Candrlic, S.; Katić, M.A.; Dlab, M.H. Online vs. Paper-based testing: A comparison of test results. In Proceedings of the 37th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), Opatija, Croatia, 26–30 May 2014; pp. 657–662. [CrossRef]
6. Jeong, H. A comparative study of scores on computer-based tests and paper-based tests. *Behav. Inf. Technol.* **2014**, *33*, 410–422. [CrossRef]
7. Santosh, K.C.; Antani, S.K. Recent trends in image processing and pattern recognition. *Multimed. Tools Appl.* **2020**, *79*, 34697–34699. [CrossRef]
8. Jocovic, V.; Marinkovic, M.; Stojanovic, S.; Nikolic, B. Automated assessment of pen and paper tests using computer vision. *Multimed. Tools Appl.* **2023**, 1–22. [CrossRef]
9. Alomran, M.; Chai, D. Automated scoring system for multiple choice test with quick feedback. *Int. J. Inf. Educ. Technol.* **2018**, *8*, 538–545. [CrossRef]
10. Fisteus, J.A.; Pardo, A.; García, N.F. Grading Multiple Choice Exams with Low-Cost and Portable Computer-Vision Techniques. *J. Sci. Educ. Technol.* **2013**, *22*, 560–571. [CrossRef]
11. Lu, M.; Zhou, W.; Ji, R. Automatic Scoring System for Handwritten Examination Papers Based on YOLO Algorithm. *J. Phys. Conf. Ser.* **2021**, *2026*, 12–30. [CrossRef]
12. Shaikh, E.; Mohiuddin, I.; Manzoor, A.; Latif, G.; Mohammad, N. Automated grading for handwritten answer sheets using convolutional neural networks. In Proceedings of the 2019 2nd International Conference on New Trends in Computing Sciences (ICTCS), Amman, Jordan, 9–11 October 2019; pp. 1–6. [CrossRef]
13. Abdul Salam, M.; El-Fatah, M.A.; Hassan, N.F. Automatic grading for Arabic short answer questions using optimized deep learning model. *PLoS ONE* **2022**, *17*, 269–272. [CrossRef]
14. Tulu, C.N.; Ozkaya, O.; Orhan, U. Automatic short answer grading with SemSpace sense vectors and MaLSTM. *IEEE Access* **2021**, *9*, 19270–19280. [CrossRef]
15. Lin, Y.; Zheng, L.; Chen, F.; Sun, S.; Lin, Z.; Chen, P. Design and Implementation of Intelligent Scoring System for Handwritten Short Answer Based on Deep Learning. In Proceedings of the 2020 IEEE International Conference on Artificial Intelligence and Information Systems (ICAIIS), Dalian, China, 20–22 March 2020; pp. 184–189. [CrossRef]
16. Sigut, J.; Castro, M.; Arnay, R.; Sigut, M. OpenCV basics: A mobile application to support the teaching of computer vision concepts. *IEEE Trans. Educ.* **2020**, *63*, 328–335. [CrossRef]
17. Huang, R.; Pedoeem, J.; Chen, C. YOLO-LITE: A real-time object detection algorithm optimized for non-GPU computers. In Proceedings of the 2018 IEEE International Conference on Big Data (Big Data), Seattle, WA, USA, 10–13 December 2018; pp. 2503–2510. [CrossRef]
18. Meir, Y.; Tevet, O.; Tzach, Y.; Hodassman, S.; Gross, R.D.; Kanter, I. Efficient shallow learning as an alternative to deep learning. *Sci. Rep.* **2023**, *13*, 5423. [CrossRef] [PubMed]
19. Zhang, J.; Yu, X.; Lei, X.; Wu, C. A novel deep LeNet-5 convolutional neural network model for image recognition. *Comput. Sci. Inf. Syst.* **2022**, *19*, 1463–1480. [CrossRef]
20. Gholami, A.; Kim, S.; Dong, Z.; Yao, Z.; Mahoney, M.W.; Keutzer, K.A. Survey of quantization methods for efficient neural network inference. *arXiv* **2021**, arXiv:2103.13630.

21. Huang, Q. Weight-quantized squeezenet for resource-constrained robot vacuums for indoor obstacle classification. *AI* **2022**, *3*, 180–193. [CrossRef]
22. Tasci, M.; Istanbullu, A.; Kosunalp, S.; Iliev, T.; Stoyanov, I.; Beloev, I. An Efficient Classification of Rice Variety with Quantized Neural Networks. *Electronics* **2023**, *12*, 2285. [CrossRef]
23. Tang, Z.; Luo, L.; Xie, B.; Zhu, Y.; Zhao, R.; Bi, L.; Lu, C. Automatic sparse connectivity learning for neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* **2022**, 1–15. [CrossRef]
24. Sher, A.; Trusov, A.; Limonova, E.; Nikolaev, D.; Arlazarov, V.V. Neuron-by-Neuron Quantization for Efficient Low-Bit QNN Training. *Mathematics* **2023**, *11*, 2112. [CrossRef]
25. Lin, S.; Ma, X.; Ye, S.; Yuan, G.; Ma, K.; Wang, Y. Toward extremely low bit and lossless accuracy in dnns with progressive admm. *arXiv* **2019**, arXiv:1905.00789.
26. Deng, L. The mnist database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Process. Mag.* **2012**, *29*, 141–142. [CrossRef]
27. Camus, L.; Filighera, A. Investigating transformers for automatic short answer grading. In Proceedings of the Artificial Intelligence in Education: 21st International Conference, AIED 2020: Proceedings, Part II 21, Ifrane, Morocco, 6–10 July 2020; pp. 43–48. [CrossRef]
28. Patil, A.; Rane, M. Convolutional neural networks: An overview and its applications in pattern recognition. In Proceedings of the Information and Communication Technology for Intelligent Systems: Proceedings of ICTIS 2020, Singapore, Singapore, 22 October 2020; pp. 21–30. [CrossRef]
29. Rehman, A.; Naz, S.; Razzak, M.I. Writer identification using machine learning approaches: A comprehensive review. *Multimed. Tools Appl.* **2019**, *78*, 10889–10931. [CrossRef]