Int J Comput Intell Syst

https://doi.org/10.1007/s44196-025-01145-3

Article in Press

Character Segmentation in Brahmi Script: Object Detection and KNN Fusion Approach

Received: 24 October 2025

Accepted: 30 December 2025

Published online: 31 January 2026

Cite this article as: Gautam N. & Chai S.S. Character Segmentation in Brahmi Script: Object Detection and KNN Fusion Approach. *Int J Comput Intell Syst* (2026). https://doi.org/10.1007/s44196-025-01145-3

Neha Gautam & Soo See Chai

Character Segmentation in Brahmi Script: Object Detection and KNN Fusion Approach

Neha Gautam[1*] and Soo See Chai[2]

[1]Faculty of Computing and Informatics, Universiti Malaysia Sabah, Kota Kinabalu, 88400, Sabah, Malaysia.
[2]Faculty of Computer Science and Information Technology, Universiti Malaysia Sarawak, Kota Samarahan, 94300, Sarawak, Malaysia.

*Corresponding author(s). E-mail(s): nehagautam1208@gmail.com;
Contributing authors: sschai@unimas.my;

Abstract

Automated word recognition has seen significant advancements, and character segmentation remains a crucial step in this process. Although several studies have focused on the segmentation of popular modern scripts, there has been relatively little attention on ancient scripts due to their infrequent use and complex structural features. Brahmi, an ancient script with a rich history, presents unique challenges for segmentation, including disconnected dot components and complex compound characters, which existing methods fail to address effectively. This study proposed a novel two-stage segmentation framework that combines contour-based object detection with a K-Nearest Neighbors (KNN)-based reattachment mechanism. The novelty of this approach lies in its ability to accurately reattach disconnected components (e.g., isolated dots) to their corresponding base characters, a problem not adequately solved in previous studies. The process comprises two main parts: line segmentation and character segmentation. The performance of the proposed approach was evaluated using printed Brahmi texts, achieving 99.81% accuracy for line segmentation and 97.32% for character segmentation, resulting in an overall average accuracy of 98.19%. These results demonstrate that the proposed hybrid framework not only surpasses prior Brahmi segmentation efforts but also provides a generalizable solution for ancient Indic scripts with similar structural challenges.

Keywords: Brahmi Script, K-nearest neighbors, Object Detection, Optical Character Recognition (OCR), Tamil Script

1 Introduction

In the text recognition domain, segmenting text images into distinct entities, such as text lines, words, and characters, is an important problem to solve [1, 2]. This task can be accomplished through either automatic segmentation methods or manual isolation of characters from the text. An automatic segmentation method is necessary for an automated recognition system [3]. The point at which the segmentation occurs is critical, as it determines how the text is separated into various entities. It is important that each entity is meaningful and accurately reflects the intended text [4]. The segmentation method used for different scripts varies based on the unique features of each script. Segmentation is particularly challenging due to the distinct features of each script and the way in which the script is written. For instance, English words have characters that are not physically connected, whereas Arabic and Devanagari scripts have physically connected characters, which poses a challenge for segmentation[4–6].

Numerous segmentation methods have been developed for scripts, such as English [7], Arabic [8], and Devanagari [9], which aid in the development of recognition systems. Similarly, various techniques have been proposed for isolating Brahmi characters from Brahmi text. However, these studies have not been successful in segmenting all types of characters. For instance, Gautam, et al. [10] were unable to isolate characters with dot (.) features. Additionally, Singh and Kushwaha [11] recently introduced a Brahmi text segmentation algorithm, but it did not achieve satisfactory performance due to the unique features of the Brahmi script. As a result, this remains an open research problem [11].

Brahmi words exhibit features similar to both English and Devanagari words. For instance, like English words, Brahmi characters are not physically connected to each other. In addition, Brahmi words have a compound character property where the "matra" is connected to the consonant, similar to the Devanagari script, Bangla script, Tamil script, Telugu script, etc. [5]. When a consonant is followed by a vowel, the shape of the consonant changes, which is referred to as "matra".

Different studies have employed diverse approaches to segment characters from standardized script texts. Pramanik and Bag [12] proposed a system that initially corrects the skew of handwritten words in Bangla and Devanagari, estimates the headline, and then segments the words into meaningful pseudo-characters. Similarly, a binary quadratic process was developed for word segmentation in English text, considering the relationship between inter-word gaps and intra-word gaps [13]. Indian scripts like Bangla [14], Devanagari [14], and Gurumukhi [15] can be segmented by dividing the text into three zones: upper, middle, and lower. The upper and lower zones focus on separating the "matra" from the word, while all characters of a word belong to the middle zone. In conclusion, a variety of studies and approaches can be observed in the segmentation of scripts across different languages worldwide.

Despite progress, existing Brahmi segmentation methods struggle with disconnected components (e.g., isolated dots), skewed characters, and compound structures. These limitations hinder accurate character segmentation in historical and derivative Indic scripts. To address this gap, we propose a two-stage segmentation framework that integrates object detection with a KNN-based reattachment mechanism. The

novelty of this work lies in its ability to reattach disconnected components using engineered positional features, a problem not adequately solved in prior studies. Unlike prior Brahmi segmentation methods that either ignored disconnected components or mis-segmented compound structures, our two-stage approach uniquely resolves these challenges, setting it apart as the first method to explicitly handle isolated dots through KNN-based reattachment.

The paper is structured in the following manner: Section two outlines the specifics of the Brahmi script, including its characteristics and properties. Section three is focused on the relevant research on text, line, and word segmentation of different texts. Section four comprehensively explains the proposed methodology for segmenting text lines. Section five presents the findings from the experiments, while Section six details the conclusions and future work.

2 Brahmi script

The Brahmi script is an ancient writing system used between the 3^{rd} century BCE and the 6^{th} century CE [16, 17]. This script was primarily used for writing religious texts related to Buddhism and Jainism [18, 19]. In the past, the Brahmi script was also referred to as the "pin-man" script, which means "stick figure" script in modern English [19, 20]. This is because the Brahmi characters are characterized by geometric features such as lines, curves, corners, and dots [21]. These features make the Brahmi script distinctive and distinguishable from other scripts, as shown in Figure 1.

Fig. 1: Character and vowels of Brahmi script [22]

2.1 *Properties of Brahmi script*

The Brahmi script, one of the earliest Indic writing systems, has distinct properties relevant to character segmentation:

1. Writing direction – Text is written from left to right, similar to Latin and Devanagari scripts.
2. Compound formation – Consonants combine with vowels to form compound characters, with modifications placed at the top, bottom, left, or right of the base consonant [23].
3. Complex compounds – Multiple consonants and vowels can merge, producing complex compound characters comparable to Devanagari and Bangla.
4. Brahmi comprises 368 characters: 33 consonants, 10 vowels, and 325 compounds [23, 24]. For example, all vowels in Figure 2 combine with the first consonant (a) from Figure 1 to form compound characters shown in Figure 3.

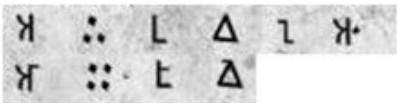


Fig. 2: Vowels of the Brahmi Script [22]

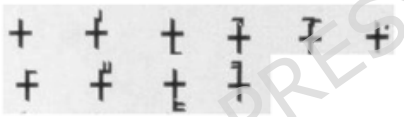


Fig. 3: Example of Consonant and Compound Characters [22]

2.2 *Characteristic of Brahmi script*

Character segmentation in Brahmi is challenging due to the following structural features:

1. Disconnected components – Certain characters include a dot (.) feature that is visually separate from the base character, often misclassified as an independent character (see Figures 3 and 4).
2. Variable matra placement – When vowels modify consonants, the resulting matra may appear on different sides of the consonant, altering its visual structure (see Figure 4).
3. Complex compounds – Multi-consonant and vowel combinations produce irregular forms, which segmentation methods may either merge into a single unit or incorrectly split (see Figures 3 and 4).

Figure 4 shows a Brahmi text sample containing individual, compound, and complex compound characters, illustrating the structural diversity that complicates segmentation.

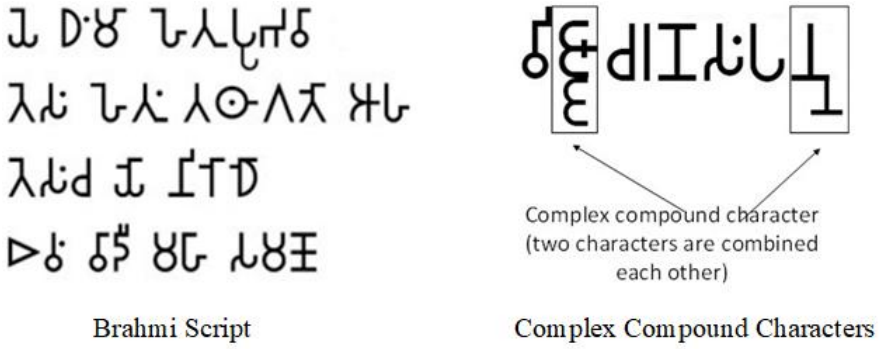


Fig. 4: Example Text in Brahmi Script and Complex Compound Characters

3 Literature

Research on text and character segmentation spans a wide range of languages and scripts, from widely used modern scripts to ancient ones. Each script presents distinct structural features, which makes segmentation a script-dependent challenge. While some studies have focused specifically on Brahmi, others have examined segmentation in related Indic scripts such as Devanagari, Bangla, and Tamil, or have explored general object detection and segmentation methods that can be adapted for character recognition tasks. To contextualize the novelty of our work, this section is organized into three parts: (i) Brahmi-specific segmentation studies, (ii) segmentation approaches applied to related Indic scripts, and (iii) general object detection and segmentation techniques.

3.1 *Brahmi Segmentation Methods*

Several research studies have explored the field of Brahmi character and word recognition. Many of these focused on isolated characters [11, 21, 24, 25], but a comprehensive recognition system requires robust segmentation of Brahmi text.

Only a few studies have specifically addressed Brahmi character and word segmentation. Gautam et al. [10] employed a segmentation method but failed to isolate characters such as i and ī due to disconnected features, limiting performance to characters with physically connected parts. Singh and Kushwaha [11] applied projection profile techniques, but performance degraded with skew variations and disconnected features. They also noted the absence of a standardized method for Brahmi segmentation.

Nagane and Mali [26] achieved high accuracy (100% identification, 99.16% segmentation) on touching characters using ensemble classification, but their method ignored

non-connected characters — a significant feature of Brahmi. In another study [25], they reported 100% line segmentation and 88.68% character segmentation, though performance was constrained by assumptions of consistent character size.

In summary, existing Brahmi approaches either fail to handle disconnected components, suffer under skew, or ignore complex compound characters. This leaves the segmentation of Brahmi text as an open research problem.

3.2 *Segmentation in Related Indic Scripts*

The Brahmi script, like English, has unconnected characters, but also includes the matra property similar to Devanagari, Bangla, Tamil, and Telugu.

For Devanagari, Srivastav and Sahu [27] reported 90% accuracy using vertical and horizontal projection profiles, while Sahare and Dhok [28] combined projection profiles with graph distance theory and SVM, achieving 97% accuracy but with a relatively high bad segmentation rate. Narang et al. [29, 30] segmented ancient Devanagari texts with projection-based methods, achieving up to 97% accuracy, but without standardized datasets and with limited focus on characters.

For Bangla, Ahmed et al. [31] used zone-based segmentation (upper, middle, lower) with CNN classification, achieving 100% line/word segmentation but leaving character segmentation underexplored. Ferdous et al. [32] studied compound characters already written separately, simplifying segmentation. Abir et al. [33] removed the matra line before isolating characters, reporting 94.32% accuracy. Pramanik and Bag [12] corrected skew and segmented pseudo-characters, reaching 94% accuracy.

For Tamil, Gayathri Devi et al. [34] introduced Text Line Slicing (TLS), which was later extended [35] for palm-leaf manuscripts, but TLS fails for characters with disconnected features. Maheswari et al. [36] developed a multi-step segmentation process (profile smoothing, over-segment reduction, etc.) for inscriptions.

Overall, while Indic segmentation methods achieve good performance, they are typically projection-based and struggle with disconnected or compound characters — the very challenges Brahmi presents.

3.3 *General Object Detection and Segmentation Approaches*

Beyond script-specific methods, several object detection and machine learning approaches have been applied to character segmentation. Preethi and Mamatha [37] used a region-based CNN for inscriptional text, achieving up to 74.79% accuracy on clean images but poor performance on noisy data. Abbass and Marhoon [38] extracted Iraqi license plate characters using contour detection, while Siddharth et al. [36] employed projections and contours for English URL extraction.

Other regional scripts have also benefited from object detection: Rajyagor and Rakholia [39] combined projection and connected components for Gujarati segmentation, and Ayyoob and Ilyas [40] applied contour detection for Vattezhuthu. Similarly, Bhatta et al. [41] used contours for license plate segmentation.

A common limitation of these general methods is their reliance on features being physically connected. While effective for Latin-based and some Indic scripts, they are not directly suitable for Brahmi, where many characters include disconnected components such as isolated dots.

Most prior Brahmi segmentation methods either ignored disconnected features or failed under skew and compound structures. Related Indic segmentation approaches, while more advanced, also rely heavily on projection-based techniques that do not generalize to Brahmi's unique challenges. General object detection methods succeed when features are physically connected, but are not robust for disconnected characters of the Brahmi script.

This gap motivates our proposed two-stage approach: contour-based object detection combined with a novel KNN-based reattachment mechanism, explicitly designed to handle disconnected elements such as dot features in Brahmi.

4 Methodology

The overall task of Brahmi and Tamil character segmentation in this study was completed through a structured pipeline Figure 5. First, input text samples were collected from the Brahmi standard dataset and the PHDIndic-11 Tamil dataset. The images were pre-processed by converting them to binary form and applying morphological operations to remove noise and unwanted objects. Next, the object detection module segmented characters without disconnected dot (.) features. Since many Brahmi and Tamil characters contain such disconnected components, a second stage was required. In this stage, a feature engineering process was applied, where positional pixel features (top-right, bottom-right, top-left, bottom-left, and centre) were extracted for each dot and candidate base character. Using these features, a k-nearest neighbor (KNN) classifier (k=1) merged each dot with its most probable base character, reattaching disconnected elements. Finally, the outputs of both stages were combined to produce the segmented characters. The system's performance was evaluated separately for object detection and KNN, and then jointly for the complete pipeline, using both accuracy and ROC–AUC metrics. Details of each part of the section is discussed further.

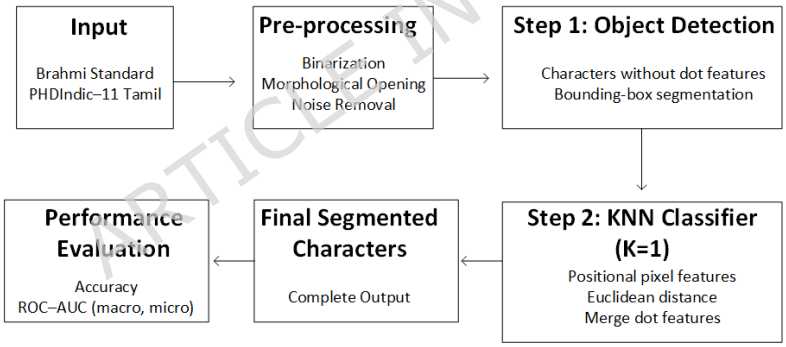


Fig. 5: Workflow of the Proposed Hybrid Character Segmentation Approach

4.1 *Pre-processing*

Thresholding is a crucial step in the pre-processing stage, aimed at improving the visibility and legibility of the input. In this study, the pre-processing phase utilizes thresholding techniques to enhance the quality of the input data.

Thresholding, also known as binarization, is a technique used to convert a grayscale image into a binary image. Its primary purpose is to enhance the visibility of edges, prioritizing the shape of regions over the intensities of pixels. There are two common approaches to thresholding: global threshold and local threshold.

In the global threshold method, a single threshold value is selected for the entire image based on the background characteristics. This approach is suitable when the image has a simple and uniform background. On the other hand, the local threshold approach involves choosing different threshold values for each pixel based on the data within a local area. This method is often used for real-life documents that may have complex and colorful backgrounds intentionally designed for stylistic purposes.

In the case of the Brahmi dataset, all images had a white background, which was not complicated. Therefore, a global threshold approach was employed in this study to perform the thresholding process

4.2 *Noise Removal*

Morphological opening is a commonly used operation in image processing to remove noise and small unwanted objects from binary or grayscale images. It is particularly effective in reducing noise while preserving the overall structure and shape of larger objects. Morphological opening involves two fundamental operations: erosion followed by dilation.

4.2a Erosion

Erosion is a morphological operation that shrinks or erodes the boundaries of objects in an image. It achieves this by scanning the image with a small structuring element, typically a small binary matrix or a kernel, and replacing each pixel in the image with the minimum value of all the pixels within the structuring element. The structuring element defines the shape and size of the erosion operation.

The erosion operation effectively erodes away the smaller details, including noise and thin structures, while preserving the larger connected components. It removes isolated pixels and breaks up narrow connections between objects.

4.2b Dilation

Dilation is the second step of morphological opening. It is the opposite of erosion and expands or dilates the boundaries of objects in an image. Like erosion, dilation also uses a structuring element and replaces each pixel in the image with the maximum value of all the pixels within the structuring element.

Dilation helps to restore the size and shape of objects while filling in gaps or holes created by erosion. It smoothens the edges and reconnects broken parts of objects, making them more complete.

This study applied erosion followed by dilation, morphological opening effectively removes small noise elements and fine details from the image while preserving the main structures and objects.

Furthermore, the study also aimed to eliminate unwanted objects or pixels by considering the area of objects. A threshold value was determined to distinguish between meaningful characters and noise. In this study, any object with an area below the threshold value, which was set at 20 pixels, was considered as noise and removed from the image.

4.3 *Segmentation*

In this study, a hybrid approach combining object detection and k-nearest neighbors (KNN) was proposed for character segmentation in Brahmi text.

4.3a Object detection

Object detection was employed as the initial step to identify all the characters within the text. The study utilized the Contour Detection approach to detect the contours of individual characters, and then the Hierarchy of characters was utilized to arrange the characters in a sequential order.

4.3b Contour Detection

Contour detection with an 8-connectivity neighbourhood is a technique used in image processing and computer vision to identify and extract the boundaries of objects or regions of interest within an image. It provides more detailed information about the object boundaries by considering diagonal connections in addition to the horizontal and vertical connections.

The explanation of the contour detection process with an 8-connectivity neighbourhood for segmentation of character from Brahmi text is further discussed.

The first step is to convert the image into binary image which is already done in pre-processing part, no need to do it again here. This process distinguishes the character of interest from the background by assigning binary values (typically 0 for background and 1 for foreground) to each pixel based on its intensity or color. Further steps of this approach is explained below:

1. Identify Foreground Pixels: Once the image is converted to binary, identify the foreground pixels representing the character or regions of interest. These are the pixels with a value of 1 in the binary image.
2. Initialize the Contour Set: Create an empty contour set to store the detected contours.
3. Traverse the Image: Iterate through each foreground pixel in the binary image.
4. Check 8-Connectivity Neighbours: For each foreground pixel, consider its 8-connectivity neighbours, which include the eight pixels surrounding the current pixel, including diagonal connections. These neighbours are indexed as follows:

Algorithm 1 Contour Detection with 8-connectivity Neighbourhood for Segmentation

Input: A binary image with height (h) and width (w)
Output: A list of contour sets

Initialize an empty list of contour sets
Initialize a 2D array or matrix "visited" of size (h, w) to keep track of visited pixels. Set all elements to false initially
Define the 8-connectivity neighborhood offsets as follows: offsets = [(-1, -1), (-1, 0), (-1, 1), (0, -1), (0, 1), (1, -1), (1, 0), (1, 1)]
Iterate over each row j from 0 to h-1:
- Iterate over each column i from 0 to w-1:
  - o **If** pixel (i, j) is equal to 1 and visited [j][i] is false, **then**:
    - → Create a new contour set, initialize it as an empty list, and append it to the list of contour sets.
    - → Call the function "traceContour (i, j)" with the current pixel coordinates.

Define the function "traceContour(x, y)" to recursively trace the contour:
- Set visited [y][x] as true.
- Add the current pixel (x, y) to the last contour set in the list of contour sets.
- Iterate over each offset in offsets:
  - o Calculate the new coordinates nx=x+offset[0] and ny=y+offset[1]
  - o **If** the new coordinates (nx, ny) are within the image boundaries and visited[ny][nx] is false, **then**
    - → **If** pixel(nx, ny) is equal to 1, call the function "traceContour(nx, ny)" recursively.
    - → **Else**, add the current pixel (x, y) to the last contour set in the list of contour sets as a boundary pixel.

Return the list of contour sets.

1 2 3
4 *P* 5
6 7 8

5. Identify Contour Points: Analyse the neighbourhood of each foreground pixel to identify contour points. A contour point is defined as a foreground pixel that has at least one background (0) neighbour. This indicates a transition from the object to the background.
6. Store Contour Points: Add each identified contour point to the contour set. This set of contour points represents the boundary or outline of the object or region of interest.
7. Repeat for All Foreground Pixels: Repeat steps 5-7 for all foreground pixels in the binary image to ensure that all contour points are detected.

By following this process, contour detection with an 8-connectivity neighbourhood allows for more precise segmentation and boundary extraction, considering both

diagonal and orthogonal connections between pixels. It captures more detailed shape information and is used to segment the characters according to the outer structure. The algorithm of counter-detection is presented below.

By using this the algorithm 1, all objects or characters can be identified.

4.3c Hierarchy of character

Contour Detection is used to extract the character from the text, and the text is written in a sequence such as; Brahmi script follows left to write, and Arabic script follows right to left. Thus, first, focus on the hierarchy of a character that should be left to right and then top to bottom.

The output of this object detection approach, it is focused on the outer edge of character, which is connected to and the connection check by 8-connectivity Neighbourhood and assign them a number in sequence. By using this approach, all objects which are physically un-touched, is considered as a character. Figure 6 and 7 explain the input image and output of object Detection part, respectively.

The analysis of both images reveals that the input image contains 34 characters, including compound and complex-compound characters. However, during the object detection process, it identified a total of 40 characters, considering each dot (.) feature as a separate character. This discrepancy arises because the object detection approach treats dot features as individual characters since they are not physically connected to the respective characters. Consequently, it can be concluded that the previous step failed to accurately detect all characters that possess dot features.

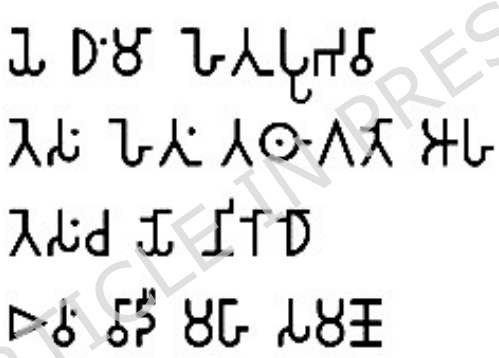


Fig. 6: Input image

Few steps were taken to address the issue of the dot (.) feature not being considered as part of the previous character in Brahmi text, which is further explained. Firstly, it was observed that the size of the dot (.) character is smaller than the regular characters, and it was determined that the size of the dot (.) should be less than 1/3rd of the average size of a character. The size of a character was calculated based on the width and height of the region of interest (ROI), as determined in the object detection stage. The area of a character was calculated using Equation (1).

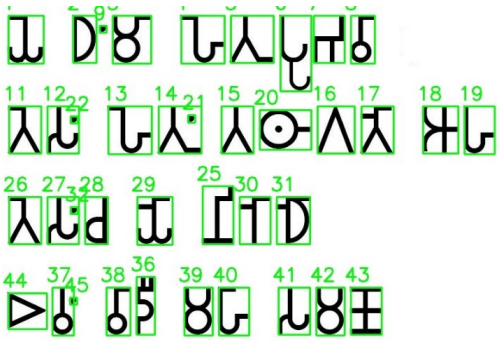


Fig. 7: Output of object detection part

$$\text{Area of a Character} = \text{Height} \times \text{Width} \tag{1}$$

By calculating the areas of all characters, the system was able to differentiate between the actual characters and the dot (.) feature.

To find out that the dot feature belongs to which character, KNN is used. KNN, which stands for k-nearest neighbours, is a supervised machine learning algorithm used for classification and regression tasks but for this study, K-NN is using for classification as this is a classification problem. The main idea behind KNN is to predict the class or value of a new instance by comparing it to the k nearest neighbours in the given dataset. In other words, KNN determines the label of a new data point based on the labels of its neighbouring data points. The number of neighbours, k, is a hyper parameter that needs to be specified before applying the algorithm. For this study the value of K is 1 as dot feature can be a part of a character only. Euclidean distance formula (Equcation 2) were considered to find out the distance between features.

$$\text{Euclidean distance} = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{2}$$

Here, x_i, y_i, and n represent the features of characters, dot (.), and the total of five features considered in the study, respectively.

4.3d Calculate distances

For a new instance in the testing dataset, the distance is calculated between that instance and each instance in the training dataset. In this study, five positional pixel features were extracted for each segmented object: *top-right, bottom-right, top-left, bottom-left, and center* pixel values. These features were selected because they effectively capture the relative position of small components (e.g., dot features) with respect to larger base characters.

For example, if a dot (.) feature appears above the top-right of a consonant, its feature vector will closely match the top-right pixel values of that consonant. Conversely, if the dot were compared with an unrelated character on the left, the Euclidean distance between their feature vectors would be significantly larger.

The distance between each dot and nearby characters was calculated using the Euclidean distance formula (Equation 2). By setting $k = 1$ in the KNN classifier, each dot was automatically merged with the closest base character. This process ensures that disconnected elements are reassigned to their correct parent characters during segmentation.

Algorithm 2 K-Nearest Neighbours (KNN) with K=1 and Euclidean Distance

Input: 1. Training dataset: X_{train} (array-like, shape [$n_{samples}$, 5]) - features of training instances
2. Y_{train} (array-like, shape [$n_{samples}$]) - class labels for training instances
3. Testing instance: X_{test} (array-like, shape [5]) - features of the testing instance

Output: Predicted class label for the testing instance: Y_{pred}

Calculate distances
- Initialize an empty list distances [ ]
- for each instance X_{train}, Y_{train} in zip(X_{train}, Y_{train}) do a. Calculate the Euclidean distance dist between X_{train} and X_{test}:
  → dist = $sqrt(sum((x - y)^2$ for x, y in zip(X_{train}, X_{test})))
  → Append (dist, Y_{train}) to distances [ ] (pairing distance with corresponding label)

Sort distances[ ] in ascending order based on distance values.
Select the nearest neighbour:
- Get the label of the nearest neighbour: Y_{pred} = distances[0][1]

Return the predicted class label Y_{pred}

4.3e Classification

For classification tasks, the class label of a new instance is usually determined based on the majority class of its k nearest neighbours. In this study, however, the objective was not general classification but the reattachment of disconnected dot (.) features to their corresponding base characters. Therefore, k was set to 1, ensuring that each dot is directly assigned to the nearest character without requiring a voting process.

The novelty of this feature engineering lies in the design of positional pixel features specifically tailored to the Brahmi script, where disconnected elements such as dot features occur frequently. Unlike generic shape or texture descriptors, the proposed feature set explicitly encodes spatial relationships between small components and their base characters. By combining these positional features with a k-nearest neighbour classifier, the system provides a simple yet effective mechanism for reattaching disconnected elements to their intended characters - a segmentation challenge not adequately addressed in previous Brahmi studies.

Although Algorithm 2 describes a basic form of the KNN algorithm with k=1, this choice was deliberate for the Brahmi segmentation task. Since each dot (.) can logically belong to only one base character, majority voting with larger values of k

would not provide additional benefit. Furthermore, using Euclidean distance on a compact set of positional features proved sufficient to capture the relative spatial relationships between dots and base characters. KNN was also chosen for its simplicity, interpretability, and suitability for low-resource settings, where more complex deep learning models may be unnecessary or impractical.

Although rule-based distance thresholds could be used for dot reattachment, such approaches require manually tuned, script- and resolution-specific parameters that do not generalize well across different fonts, character sizes, or layouts. In contrast, the KNN-based formulation provides a data-driven distance-based decision mechanism that adapts naturally to spatial variations without relying on handcrafted rules. Moreover, unlike centroid-only nearest assignment, the proposed KNN approach incorporates multiple positional pixel features, allowing it to capture asymmetric spatial relationships (e.g., top-right versus bottom-left dot placement), which are characteristic of Brahmi compound characters.

The algorithm of KNN is represented in Algorithm 2. After applying the KNN step, all dot (.) features were successfully merged with their corresponding base characters. The resulting segmentation output after KNN reattachment is illustrated in Figure 8, while Figure 9 shows the final segmented characters saved as individual entities.

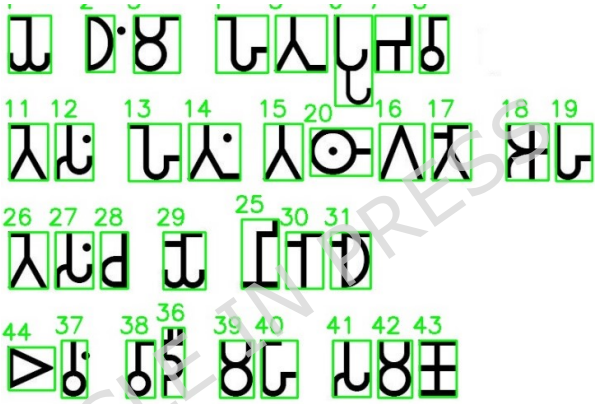


Fig. 8: After merging the dot character with relevant character

All of the characters that have been defined by their ROI, are now regarded as individual characters and are saved. The segmented character can be seen in Figure 9.

5 Results

The methodology employed in this study consisted of two parts: object detection and KNN, which were executed sequentially. Experiments were carried out on two datasets: the Brahmi standard dataset [42] and the PHDIndic-11 dataset [43].

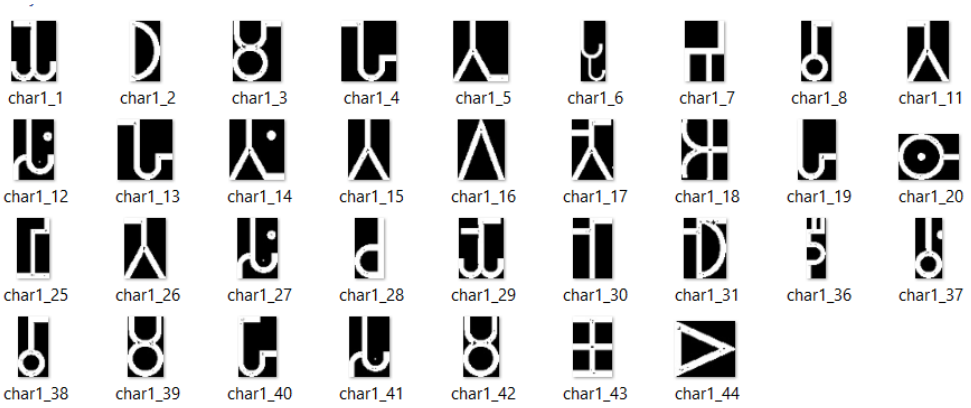


Fig. 9: Example of all segmented character

5.1 Results on Brahmi Script

The Brahmi dataset [42] consisted of 55 text samples with a total of 663 characters, including 49 compound characters and 112 characters with dot (.) features. The object detection approach was applied only to characters without dot features due to the limitations of this stage. Out of the 551 characters tested (663 total characters minus 112 dot-feature characters), 550 were segmented correctly. Errors occurred primarily in complex compound characters, which were sometimes treated as two separate objects. Out of 49 complex compound characters, 48 were correctly segmented, while one was segmented correctly in only one of its two occurrences. These errors were caused by the difficulty of recognizing connecting strokes, leading to incorrect splitting. An example of segmented characters is illustrated in Figure 9.

The KNN model was then evaluated specifically for dot (.) features. The Brahmi dataset contained 6,460 training characters and 1,360 testing characters. For dot-feature evaluation, 250 training and 75 testing samples were used. Out of 75 unseen testing samples with dot features, 73 were correctly recognized, yielding an accuracy of 97.33% using Equation 3.

$$Accuracy = \frac{\text{Number of correctly segmented characters}}{\text{Total number of characters}} \times 100 \tag{3}$$

After evaluating each model separately, both approaches were combined and tested on all 55 text samples containing 663 characters. The system incorrectly segmented 12 characters in total (1 in the object detection stage and 11 in the KNN stage), resulting in an overall segmentation accuracy of 98.19%.

5.2 Results on Tamil Script

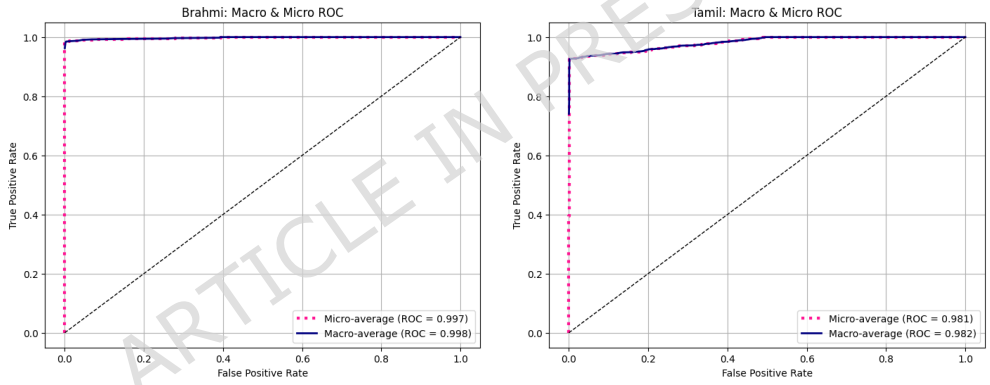
To evaluate the adaptability of the proposed method to other ancient scripts, experiments were also performed on the PHDIndic-11 dataset [43], consisting of handwritten

Tamil text. This dataset poses additional challenges due to handwriting variations, inconsistent stroke thickness, and skew, all of which reduce the reliability of projection-based methods. A total of 120 text samples were available, from which 20 were selected for analysis. Of these, 8 were used for object detection testing, 10 for extracting dot-feature characters, and 10 for system-level evaluation.

The testing dataset consisted of 1,082 characters without dot features and 536 characters with dot features. On characters without dot features, the object detection stage achieved 92.14% accuracy. On characters with dot features, the KNN stage achieved 96.82% accuracy. For overall evaluation, 10 Tamil text samples with 1,482 characters were tested. Of these, 1,337 were segmented correctly, resulting in an overall accuracy of 90.21%.

5.3 ROC–AUC Analysis

To further evaluate performance, ROC–AUC curves were computed using a one-vs-rest strategy for all 170 classes. Macro-average and micro-average ROC curves were calculated to summarize performance across classes. The micro-average aggregates contributions of all classes to evaluate overall performance, while the macro-average gives equal weight to each class, including minority ones. As shown in Figure 10, the Brahmi dataset achieved an AUC of 0.997 (micro-average) and 0.998 (macro-average), while the Tamil dataset achieved 0.981 (micro-average) and 0.982 (macro-average). These high AUC values indicate strong discriminative ability across classes, complementing the accuracy metrics reported earlier.



(a) Brahmi ROC (b) Tamil ROC

Fig. 10: Macro- and micro-average ROC curves for (a) Brahmi and (b) Tamil datasets.

5.4 Ablation Study on Dot Reattachment Strategies

One of the key contributions of this work is the reattachment of disconnected dot (.) components to their correct Brahmi characters using a KNN-based approach. To

Table 1: Ablation Study on Dot Reattachment Methods (Brahmi Script)

Reattachment Method	**Feature Representation**	**Accuracy (%)**	**Observed Limitation**
Rule-based distance threshold	Dot-to-character Euclidean distance	94.67	Performance changes with spacing and font size
Centroid-based nearest assignment	Character centroid coordinates	95.82	Incorrect assignment in dense text
Proposed KNN (k = 1)	Positional pixel features (TR, BR, TL, BL, Center)	97.33	Rare ambiguity in complex compounds

justify the choice of KNN, this study compared it with simpler distance-based reattachment methods. The goal of this comparison was to check whether basic distance rules are sufficient for dot reattachment or whether the KNN-based method provides better and more reliable results (Table 1). Three dot reattachment methods were evaluated using the same detected dot components and the same candidate base characters:

1. Rule-based distance threshold: – A dot is attached to the nearest character only if the Euclidean distance between them is below a manually chosen threshold.
2. Centroid-based nearest assignment – A dot is attached to the character whose centroid is closest to the dot, based only on Euclidean distance.
3. Proposed KNN-based reattachment (k = 1) – A dot is attached to the nearest character using KNN with five positional pixel features (top-right, bottom-right, top-left, bottom-left, and center).

The results demonstrate that while simple distance-based heuristics provide reasonable performance, they are highly sensitive to character spacing and layout variations (Table 1). In contrast, the proposed KNN-based approach consistently achieves higher accuracy by leveraging multiple positional relationships rather than relying on a single distance metric.

5.5 Comparison with Previous Studies

Table 2 compares the performance of the proposed method with existing Brahmi character segmentation techniques. Previous approaches such as projection profiles [10, 11] and connected components [26] performed reasonably well but struggled with disconnected characters, skewed text, or compound forms. The proposed method, by combining object detection with KNN reattachment, achieved 98.19% accuracy on Brahmi and 90.21% on Tamil, demonstrating superior robustness in handling both disconnected elements and complex compounds.

5.6 Discussion

Compared to prior efforts, the proposed approach demonstrates superior robustness in handling disconnected elements and complex compound characters. While projection-based methods [10, 11] and connected component techniques [26] achieved reasonable performance, they either failed to segment characters with dot features or ignored

Table 2: Comparison of segmentation performance between the proposed method and previous studies

Study	Script	Method	Accuracy (%)	Limitations
Gautam et al. (2016) [10]	Brahmi	Projection-based	Not reported	Failed to segment disconnected characters (dot features)
Singh & Kushwaha (2019) [11]	Brahmi	Projection profile	88.68	Poor performance for skewed or compound characters
Nagane & Mali (2021) [25]	Brahmi	Ensemble + connected component	99.16 (touching only)	Ignored non-connected characters
Proposed Approach (This Study)	Brahmi	Object Detection + KNN Fusion	**98.19**	Slight drop in accuracy for complex compound characters

non-connected components altogether. By combining object detection with a KNN-based reattachment mechanism, the system achieved state-of-the-art performance for printed Brahmi script and demonstrated adaptability to handwritten Tamil script.

These results confirm that the proposed approach is not limited to Brahmi text alone but can be effectively extended to other scripts with similar structural properties. Since Tamil script evolved from Brahmi and exhibits comparable features such as compound characters and disconnected components, the successful segmentation of Tamil text validates the generalizability of the method. This suggests that the framework can be applied to other Brahmi-derived or structurally similar ancient scripts (e.g., Grantha, Vattezhuthu, Kadamba) for effective character segmentation.

5.7 Qualitative Error Analysis

To provide qualitative insight into the limitations of the proposed segmentation framework, representative examples of segmentation errors are presented in this subsection. These examples illustrate situations in which the algorithm faces difficulties and help clarify the sources of segmentation errors.

Figure 11 presents a compound character example shown at two stages of the pipeline. The left image shows the output of the contour-based object detection stage, where the dot (.) feature is incorrectly detected as an independent character due to its physical disconnection from the base character. The right image shows the output after applying the KNN-based reattachment, where the dot feature is successfully merged with its corresponding compound character. This example highlights the limitation of contour-based object detection in handling disconnected components and demonstrates the effectiveness of the proposed KNN-based reattachment mechanism in correcting such errors.

Figure 12 illustrates a failure case of the KNN-based reattachment stage. In this example, multiple neighboring characters are spatially close to the dot component, resulting in similar positional relationships. Under such dense layouts, the positional pixel features of adjacent characters become ambiguous, leading to incorrect nearest-neighbor assignment and misattachment of the dot feature.

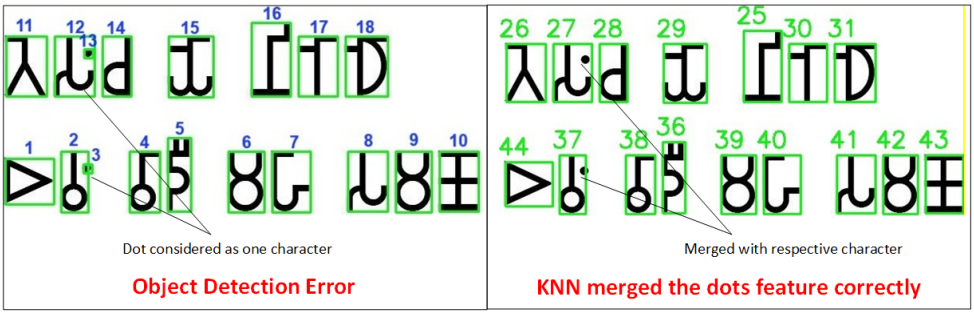


Fig. 11: Dot reattachment example. Left: output of contour-based object detection where the dot (.) is detected as a separate component. Right: result after KNN-based reattachment, where the dot is correctly merged with the compound character)

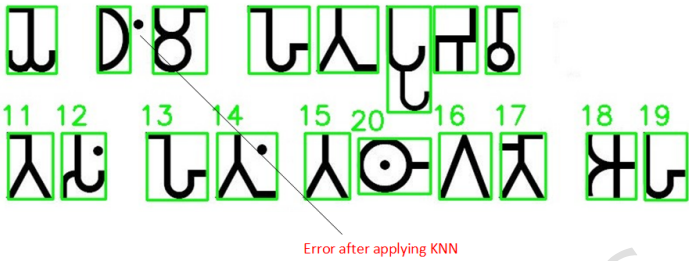


Fig. 12: Failure case of KNN-based dot reattachment in dense layouts, where the dot (.) is incorrectly merged due to spatial ambiguity among neighboring characters.

These failure cases indicate that segmentation accuracy is influenced by both character density and the clarity of structural connections. While the proposed KNN-based approach effectively resolves most disconnected dot cases, ambiguous spatial configurations remain challenging. Addressing these issues may require incorporating additional contextual information or stroke continuity cues in future extensions of the framework.

5.8 Discussion on Generalizability and Challenging Conditions

The generalizability of the proposed segmentation framework was evaluated across printed Brahmi and handwritten Tamil scripts, which exhibit distinct visual characteristics, writing styles, and degradation patterns. The successful application of the same pipeline to both datasets demonstrates that the approach is not tightly coupled to a single script or writing modality.

However, it is acknowledged that the current evaluation does not fully capture the severity of degradation commonly observed in historical manuscripts, such as ink bleed-through, surface erosion, or heavy noise. While prior studies on degraded Brahmi text have reported significant performance drops, the present framework is expected to remain effective under moderate degradation due to its reliance on contour-based detection and relative positional features rather than projection profiles.

Future work will focus on extending the evaluation to degraded historical documents, including artificially degraded samples and small-scale testing on related Brahmi-derived scripts such as Grantha or Kadamba, to further validate the robustness and generalizability of the method.

5.9 Limitations and Future Work

This study focused on printed Brahmi text due to dataset availability, which may not fully capture the challenges posed by degraded historical manuscripts. Moreover, although an ablation study was conducted to compare the proposed KNN-based reattachment with simpler distance-based alternatives, the comparison was limited to lightweight heuristics. Exploring more sophisticated learning-based reattachment strategies under severe degradation conditions remains an important direction for future research. Extending this work to degraded historical texts, testing more complex compound structures, and evaluating alternative reattachment models are also important future directions.

6 Conclusion

This study aimed to segment characters from Brahmi and Tamil texts using a two-stage hybrid approach that integrates contour-based object detection with a KNN-based reattachment mechanism. The segmentation pipeline consisted of four main steps: input, pre-processing for noise removal, segmentation, and output generation. Pre-processing involved binarization, morphological opening (erosion and dilation), and elimination of unwanted small objects. The segmentation phase employed object detection to isolate characters without dot (.) features, while KNN was used to reattach disconnected dot features to their corresponding base characters.

The proposed approach was evaluated separately for object detection and KNN, as well as jointly, on printed Brahmi and handwritten Tamil texts. For printed Brahmi text, object detection achieved 99.81% accuracy, while KNN achieved 97.33%, resulting in an overall segmentation accuracy of 98.19%. For handwritten Tamil text, object detection achieved 92.14% accuracy and KNN 96.82%, with an overall segmentation accuracy of 90.21%. These results demonstrate that the hybrid approach is robust across different datasets and capable of handling both disconnected and compound characters more effectively than previous methods.

The unique contribution of this work lies in explicitly addressing the challenge of disconnected components in Brahmi characters - particularly isolated dots - through the use of engineered positional pixel features combined with KNN-based reattachment. Unlike prior approaches that either ignored disconnected features or mis-segmented compound structures, this study provides the first systematic framework to reattach such components reliably.

Nevertheless, this work has certain limitations. The experiments were conducted primarily on printed Brahmi text due to dataset availability, which may not fully capture the variability of degraded manuscripts. Moreover, while KNN was chosen for its simplicity, interpretability, and suitability for low-resource settings, no ablation study was conducted to compare it against alternative merging strategies such as

rule-based or CNN-based approaches. Addressing these aspects forms an important direction for future research.

As future work, the proposed method may be extended to test multiple complex compound characters of Brahmi, which could not be fully explored due to limited samples in this study. Broader evaluation on diverse Tamil datasets is also encouraged. Beyond Brahmi and Tamil, the hybrid method can be adapted to other ancient Indic scripts that share similar structural properties, particularly those with disconnected elements and compound characters. Scripts such as Telugu, Devanagari, and Malayalam, which originated from or were influenced by Brahmi, could particularly benefit from the proposed framework.

7 Declarations

Ethics approval and consent to participate. Not applicable.

Consent for publication. Not applicable.

Availability of data and materials. The data supporting the findings of this study are available from the corresponding author upon reasonable request.

Competing interests. The authors declare that they have no competing interests.

Funding. This research did not receive any specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

Authors' contributions. N.G. (Neha Gautam) conceived and carried out the research work, including data collection, methodology design, experimentation, analysis, and manuscript preparation. S.S.C. (Soo See Chai) provided continuous supervision, insightful guidance throughout the research process, and critical feedback that enhanced the technical quality and clarity of the manuscript. Both authors reviewed and approved the final version of the paper.

Acknowledgements. The authors would like to thank Faculty of Computer Science & Information Technology (FCSIT), Universiti Malaysia Sarawak for their valuable support.

References

[1] Toiganbayeva N, Kasem M, Abdimanap G, Bostanbekov K, Abdallah A, Alimova A, et al. KOHTD: Kazakh offline handwritten text dataset. Signal Processing: Image Communication. 2022;108:116827. https://doi.org/10.1016/j.image.2022.116827.

[2] Mitra P, Bhattacharjee K, Das A, Dey SK, Chakraborty D, Ghosal A, et al. Character segmentation for handwritten Bangla words using image processing. American Journal of Electronics and Communication. 2021;1(3):8–11.

[3] Qiu Q, Tan Y, Ma K, Tian M, Xie Z, Tao L. Geological symbol recognition on geological map using convolutional recurrent neural network with augmented data. Ore Geology Reviews. 2022;p. 105262. https://doi.org/10.1016/j.oregeorev.2022.105262.

[4] Faizullah S, Ayub MS, Hussain S, Khan MA. A Survey of OCR in Arabic Language: Applications, Techniques, and Challenges. Applied Sciences. 2023;13(7):4584. https://doi.org/10.3390/app13074584.

[5] Sharma R, Kaushik B. Offline recognition of handwritten Indic scripts: A state-of-the-art survey and future perspectives. Computer Science Review. 2020;38:100302. https://doi.org/10.1016/j.cosrev.2020.100302.

[6] Inuzuka MA, Rocha AS, Nascimento HAD. Segmentation of words written in the Latin alphabet: a systematic review. In: Computational Processing of the Portuguese Language: 14th International Conference, PROPOR 2020, Evora, Portugal, March 2–4, 2020, Proceedings 14. Springer; 2020. p. 291–302.

[7] Kaur I, Singh KJ. Printed text recognition system for multi-script image. International Journal of Signal Processing Systems. 2016;4(5):411–416. https://doi.org/10.18178/ijsps.4.5.411-416.

[8] Ghaleb H, Nagabhushan P, Pal U. Segmentation of offline handwritten Arabic text. In: 2017 1st International Workshop on Arabic Script Analysis and Recognition (ASAR). IEEE; 2017. p. 41–45.

[9] Jindal K, Kumar R. A new method for segmentation of pre-detected Devanagari words from the scene images: Pihu method. Computers & Electrical Engineering. 2018;70:754–763. https://doi.org/10.1016/j.compeleceng.2017.12.017.

[10] Gautam N, Sharma RS, Hazrati G. Handwriting recognition of Brahmi script (an artefact): base of PALI language. In: Proceedings of First International Conference on Information and Communication Technology for Intelligent Systems: Volume 2. Springer; 2016. p. 519–527.

[11] Singh AP, Kushwaha AK. Analysis of Segmentation Methods for Brahmi Script. DESIDOC Journal of Library and Information Technology. 2019;39(2). https://doi.org/10.14429/djlit.39.2.13615.

[12] Pramanik R, Bag S. Segmentation-based recognition system for handwritten Bangla and Devanagari words using conventional classification and transfer learning. IET Image Processing. 2020;14(5):959–972. https://doi.org/10.1049/iet-ipr.2019.0208.

[13] Sharma MK, Dhaka VS. Segmentation of handwritten words using structured support vector machine. Pattern Analysis and Applications. 2020;23:1355–1367. https://doi.org/10.1007/s10044-019-00843-x.

[14] Ghosh R, Vamshi C, Kumar P. RNN based online handwritten word recognition in Devanagari and Bengali scripts using horizontal zoning. Pattern Recognition. 2019;92:203–218. https://doi.org/10.1016/j.patcog.2019.03.030.

[15] Kaur RP, Kumar M, Jindal MK. Newspaper text recognition of Gurumukhi script using random forest classifier. Multimedia Tools and Applications. 2020;79:7435–7448. https://doi.org/10.1007/s11042-019-08365-8.

[16] Seland EH. Archaeology of trade in the western Indian Ocean, 300 BC–AD 700. Journal of Archaeological Research. 2014;22:367–402. https://doi.org/10.1007/s10814-014-9075-7.

[17] Singh U. A History of ancient and Early medieval India: From the stone age to the 12th Century. India: Pearson Education; 2009.

[18] Subba JR. Evaluation of man and the modern society in Sikkim. India: Gyan Publishing House; 2008.

[19] Salomon R. Indian epigraphy: A guide to the study of inscriptions in Sanskrit, Prakrit, and the other Indo-Aryan languages. United States of America (USA): Oxford University Press; 1998.

[20] Keay J. India: A history. Atlantic: Grove Press; 2011.

[21] Gautam N, Chai SS. Optical character recognition for Brahmi script using geometric method. Journal of Telecommunication, Electronic and Computer Engineering (JTEC). 2017;9(3-11):131–136.

[22] Roy A, Mandal M. Brahmi: rediscovering the lost script. India: Resurrect Books; 2016.

[23] Pal U, Chaudhuri B. Indian script character recognition: a survey. pattern Recognition. 2004;37(9):1887–1899. https://doi.org/10.1016/j.patcog.2004.02.003.

[24] Siromoney G, Chandrasekaran R, Chandrasekaran M. Machine recognition of Brahmi script. IEEE Transactions on Systems, Man, and Cybernetics. 1983;SMC-13(4):648–654. https://doi.org/10.1109/TSMC.1983.6313155.

[25] Nagane AS, Mali S. Segmentation of Characters from Degraded Brahmi Script Images. In: Applied Computer Vision and Image Processing: Proceedings of International Conference on Computing in Engineering and Technology (ICCET) 2020. Advances in Intelligent Systems and Computing, Volume 1. Springer; 2020. p. 326–338.

[26] Nagane AS, Mali SM. Identification and Segmentation of Touching Brahmi Characters from Degraded Digital Estampage Images using Ensemble Classifier. Indian Journal of Computer Science and Engineering (IJCSE). 2021;12(6):1722–1733. https://doi.org/10.21817/indjcse/2021/v12i6/211206110.

[27] Srivastav A, Sahu N. Segmentation of Devanagari handwritten characters. International Journal of Computer Applications. 2016;142(14). https://doi.org/10.5120/ijca2016909994.

[28] Sahare P, Dhok SB. Robust character segmentation and recognition schemes for multilingual Indian document images. IETE Technical Review. 2019;36(2):209–222. https://doi.org/10.1080/02564602.2018.1450649.

[29] Narang SR, Jindal M, Kumar M. Line segmentation of Devanagari ancient manuscripts. Proceedings of the national academy of sciences, India section A: physical sciences. 2020;90:717–724. https://doi.org/10.1007/s40010-019-00627-2.

[30] Narang SR, Jindal MK, Kumar M. Drop flow method: an iterative algorithm for complete segmentation of Devanagari ancient manuscripts. Multimedia Tools and Applications. 2019;78:23255–23280. https://doi.org/10.1007/s11042-019-7620-6.

[31] Ahmed T, Raihan MN, Kushol R, Salekin MS. A complete Bangla optical character recognition system: An effective approach. In: 2019 22nd International Conference on Computer and Information Technology (ICCIT). IEEE; 2019. p. 1–7.

[32] Ferdous J, Karmaker S, Rabby ASA, Hossain SA. Matrivasha: A multipurpose comprehensive database for Bangla handwritten compound characters. In: Emerging Technologies in Data Mining and Information Security: Proceedings of International Conference on Emerging Technologies in Data Mining and Information Security (IEMIS) 2020, Volume 3. Springer; 2021. p. 813–821.

[33] Abir ASM, Rahman S, Ellin S, Farzana M, Manik MH, Rahman CR. Confronting the constraints for optical character segmentation from printed Bangla text image. In: Proceedings of the 7th International Conference on Networking, Systems and Security; 2020. p. 36–44.

[34] Gayathri Devi S, Vairavasundaram S, Teekaraman Y, Kuppusamy R, Radhakrishnan A. A Deep Learning Approach for Recognizing the Cursive Tamil Characters in Palm Leaf Manuscripts. Computational Intelligence and Neuroscience. 2022;2022. https://doi.org/10.1155/2022/3432330.

[35] Sathik MM, Ratheash RS. Text Line Segmentation in Tamil Language Palm Leaf Manuscripts–A Novel Approach. Journal of Tianjin University Science and Technology. 2021;54(4):297–304. https://doi.org/10.17605/OSF.IO/8DWSQ.

[36] Maheswari PU, Aswathy A, Ezhilarasi S, Priya MR. Recognition of Vowels, Consonants and Compound Character Sequences from Ancient Tamil Stone Inscriptions using Deep Neural Networks. In: 2022 IEEE International Power and Renewable Energy Conference (IPRECON). IEEE; 2022. p. 1–6.

[37] Preethi P, Mamatha HR. Region-Based Convolutional Neural Network for Segmenting Text in Epigraphical Images. Artificial Intelligence and Applications. 2023;1(2):119–127.

[38] Abbass GY, Marhoon AF. Car license plate segmentation and recognition system based on deep learning. Bulletin of Electrical Engineering and Informatics. 2022;11(4):1983–1989. https://doi.org/10.11591/eei.v11i4.3434.

[39] Rajyagor B, Rakholia R. Tri-level handwritten text segmentation techniques for Gujarati language. Indian Journal of Science and Technology. 2021;14(7):618–627. https://doi.org/10.17485/IJST/v14i7.2146.

[40] Ayyoob M, Ilyas PM. Efficient Binarization of Ancient Handwritten Vattezhuthu Documents. In: 2022 3rd International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT). IEEE; 2022. p. 1–3.

[41] Bhatta S, Srivastava H, Das SK, Singh P. License plate detection for smart parking management. In: Advances in Power Systems and Energy Management: Select Proceedings of International Conference on Emerging Trends and Advances in Electrical Engineering and Renewable Energy (ETAEERE) 2020. Springer; 2021. p. 71–79.

[42] Gautam N, Chai SS, Gautam M. The Dataset for Printed Brahmi Word Recognition. In: Micro-Electronics and Telecommunication Engineering: Proceedings of 3rd International Confrence on Micro-Electronics and Telecommunication Engineering (ICMETE) 2019. Lecture Notes in Networks and Systems, vol 106. Springer; 2020. p. 125–133.

[43] Obaidullah SM, Halder C, Santosh K, Das N, Roy K. PHDIndic_11: page-level handwritten document image dataset of 11 official Indic scripts for script identification. Multimedia Tools and Applications. 2018;77:1643–1678. https://doi.org/10.1007/s11042-017-4373-y.