International Journal Software Engineering and Computer Science (IJSECS)
5 (2), 2025, 498-509
Published Online August 2025 in IJSECS (http://www.journal.lembagakita.org/index.php/ijsecs)
P-ISSN: 2776-4869, E-ISSN: 2776-3242. DOI: https://doi.org/10.35870/ijsecs.v5i2.4045.

RESEARCH ARTICLE Open Access

Image Segmentation of East OKU Script Using the Bounding Box Method for Cultural Heritage Digitization

M. Fikri
Intelligent Systems Research Group (ISRG), Faculty of Science Technology, Universitas Bina Darma, Palembang City, South Sumatra Province, Indonesia.
Email: mfikri202020@gmail.com.

Ilman Zuhri Yadi *
Intelligent Systems Research Group (ISRG), Faculty of Science Technology, Universitas Bina Darma, Palembang City, South Sumatra Province, Indonesia.
Corresponding Email: ilmanzuhriyadi@binadarma.ac.id.

Yesi Novaria Kunang
Intelligent Systems Research Group (ISRG), Faculty of Science Technology, Universitas Bina Darma, Palembang City, South Sumatra Province, Indonesia.
Email: yesinovariakunang@binadarma.ac.id.

Leon Andretti Abdillah
Intelligent Systems Research Group (ISRG), Faculty of Science Technology, Universitas Bina Darma, Palembang City, South Sumatra Province, Indonesia.
Email: leon.abdillah@binadarma.ac.id.

Received: April 15, 2025; Accepted: June 20, 2025; Published: August 1, 2025.

Abstract: East Ogan Komering Ulu (OKU) is distinguished by its cultural heritage, which encompasses historical artifacts such as traditional houses, crafts, and ceremonial dances. Among the most significant cultural assets are relics inscribed with ancient scripts, including Pallawa and Ulu, which offer valuable insight into the region's historical literacy. The present study addresses the segmentation of OKU Timur script images through the Bounding Box method. This approach was selected based on its practicality and efficiency, particularly in the context of datasets where script characters exhibit straightforward forms and the overall data volume remains manageable. The segmentation process utilizes Python within the Google Colaboratory platform, ensuring accessible and reproducible workflows. Accurate segmentation is essential to support ongoing digitization and preservation of cultural scripts. The methodology involves gathering data from local artifacts, converting images to binary format, and isolating characters using Bounding Boxes. The results demonstrate that the method effectively separates individual script characters, laying the groundwork for dataset development and subsequent image classification tasks.

Keywords: OKU Timur Script; Bounding Box; Image Segmentation; Python; Google Colaboratory; Dataset.

1. Introduction

East Ogan Komering Ulu (OKU) Timur is recognized for its extensive cultural heritage, rooted in both tangible artifacts and intangible traditions. The region's legacy can be observed in its traditional dwellings, intricate craftwork, and ritual dances, all of which have persisted through successive generations. Such continuity is not merely a reflection of cultural pride, but also a testament to the community's commitment to maintaining links with its historical roots. Among the most valuable remnants of the past are artifacts inscribed with scripts such as Pallawa and Ulu. These objects, whether carved in stone, etched in wood, or painted on cloth, bear witness to the development of literacy and record-keeping practices that once flourished in the area.

The presence of these scripts points to a sophisticated understanding of language as a visual system. Writing, in this context, is more than a means of documentation; it is an instrument for expressing collective memory and social values. Through the act of inscribing language onto diverse materials, earlier generations left behind a record that extends beyond oral tradition [1]. The script native to OKU Timur, originating from South Sumatra, stands out for its distinctive visual style and its role in shaping local identity. The preservation of such writing systems is not only about safeguarding physical artifacts; it is equally about ensuring that the knowledge and cultural significance embedded within them are not lost to time. In recent years, the rapid advancement of digital technology has opened new avenues for cultural preservation. The application of computational methods to the study of ancient scripts offers an opportunity to document, analyze, and disseminate information in ways that were previously unattainable. Among these methods, image segmentation has emerged as a practical solution for isolating and studying individual characters from historical manuscripts and inscriptions. Segmentation, defined as the process of distinguishing objects from their backgrounds, is an essential step in preparing visual data for further analysis or archival purposes [2].

The research described here adopts the Bounding Box method to segment characters in images of the OKU Timur script. This approach was selected after careful consideration of its efficiency and suitability for scripts with relatively uncomplicated forms. The method enables rapid annotation, which is especially valuable when dealing with large datasets or when resources are limited. In addition, the straightforward nature of bounding box annotation makes it accessible to researchers and practitioners who may not have extensive experience with more complex segmentation techniques. By framing each character within a rectangular boundary, the method facilitates the precise localization and separation of script elements from surrounding visual noise. The significance of this approach extends beyond technical convenience. By enabling the systematic segmentation of characters, the research lays the groundwork for the creation of comprehensive digital archives. Such resources can support a range of activities, from linguistic analysis and script revitalization to educational initiatives aimed at raising awareness of local heritage. Moreover, the digitization of traditional scripts using accessible computational tools helps bridge the gap between historical legacy and contemporary scholarship, ensuring that cultural knowledge remains available to future generations. The segmentation of OKU Timur script characters through the Bounding Box method represents a strategic step toward the digital preservation of regional heritage. The outcomes of this work are expected to facilitate further research and support ongoing efforts to document and sustain Indonesia's diverse script traditions. The integration of technological methods with heritage preservation reflects an evolving approach to cultural stewardship—one that values both innovation and respect for the past.

2. Related Work

Mathew *et al.* (2015) and Memon *et al.* (2020) emphasize the urgency of documenting and digitizing indigenous scripts in response to the ongoing decline in linguistic diversity [8][9]. Their work underscores the foundational role of manual transcription and photographic documentation in cultural preservation, while also highlighting the inherent limitations of these approaches—particularly in terms of scalability and accessibility. As the volume and complexity of script artifacts grow, such traditional methods struggle to keep pace with the demands of comprehensive preservation efforts. Memon *et al.* (2020) note a significant shift in the field with the adoption of Optical Character Recognition (OCR) systems, which have enabled more efficient processing of written materials [9]. However, as Kaur and Sagar (2023) point out, the bulk of OCR research and development has centered on scripts with well-established digital resources, such as Latin and Arabic, leaving many regional scripts underrepresented in technological advances [7]. This gap has prompted researchers to explore alternative strategies for script documentation and analysis.

Kantorov *et al.* (2016) and Papadopoulos *et al.* (2016) introduced bounding box annotation as a practical tool for isolating textual elements from complex backgrounds [6][10]. Their findings show that these computer vision techniques can be adapted to various character recognition scenarios, including those involving scripts with limited prior study. When applied to regional scripts, such as Javanese and Balinese, bounding box

segmentation has been shown by Budiman *et al.* (2023) and Rasyidi *et al.* (2021) to enhance the accuracy of character classification, supporting more reliable digital archiving and analysis [3][11]. Despite these improvements, researchers continue to face obstacles when working with scripts characterized by irregular shapes, overlapping glyphs, or significant degradation. Sapitri *et al.* (2023) highlight the potential of deep learning models, particularly Convolutional Neural Networks (CNNs), to address some of these challenges by increasing recognition robustness [12]. However, the effectiveness of such models is often constrained by the need for extensive annotated datasets—a resource that is rarely available for lesser-known scripts.

Darma (2018, 2019) draws attention to the lack of tailored digital segmentation and annotation techniques for local scripts, arguing that the adaptation of bounding box methods can help bridge this gap by providing accessible and accurate solutions [4][5]. Through the integration of established computer vision strategies, research in this area is positioned to make meaningful contributions to the preservation and revitalization of Indonesia's literary heritage. While significant progress has been made in applying computational approaches to script analysis, the literature continues to reveal a shortage of scalable, accessible solutions for the digital preservation of regional scripts. The adoption of bounding box techniques—alongside ongoing innovation in machine learning and data augmentation—offers a promising direction for future work in this field.

3. Research Method

In this study on the segmentation of OKU Timur script images, the Bounding Box method is employed. The Bounding Box technique is used to mark objects that have been grouped during the object segmentation process. Marked objects are highlighted with green boxes. The methodological steps used in this research are as follows:

1) Data Collection
2) Line Segmentation
3) Character Segmentation using Bounding Box
4) Image Saving

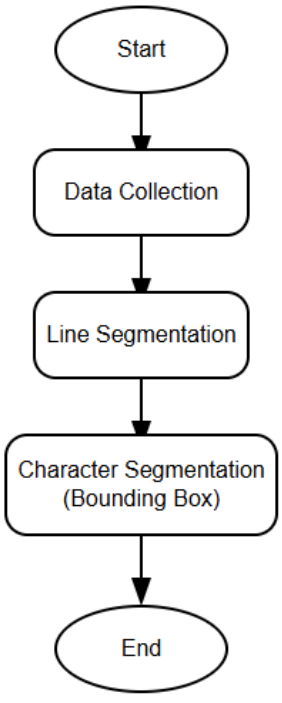

Figure 1. Flow Chart

The methodological steps for image segmentation are carried out as follows:

1) Data Collection
   The initial stage of this research involves collecting data from script artifacts found in the OKU Timur region. These artifacts were processed by researchers involved in the ISRG research group for the study titled "Development of a Mobile-Based Transliteration Application for the Ulu Script Variant of OKU Timur." The OKU Timur script was then compiled into a questionnaire to be processed in the Image Preprocessing stage, which was conducted by another ISRG team. There were 102 respondents, with each questionnaire

containing 225 OKU Timur script characters. Below is an example of a questionnaire page completed by a respondent.

Figure 2. Example of the OKU Timur Script Questionnaire

2) Segmentation
Segmentation is a crucial part of image analysis, as this procedure involves analyzing the desired image for subsequent purposes, such as pattern recognition [13]. Segmentation allows each object in the image to be taken separately, enabling them to be used as input for other processes [14]. Segmentation is used in human face recognition to distinguish the human face from the background or other body parts, resulting in a face image that can be recognized. It is also used in object type recognition to differentiate each object from the background, ensuring that the background is not processed during the recognition process. Similarly, in letter recognition within text, segmentation is necessary to identify the letters to be recognized.

3) Line Segmentation
After the Image Preprocessing stage, the resulting binary image facilitates the identification and extraction of text lines, as each text line can now be recognized as a cluster of black pixels isolated against a white background. By applying techniques such as connected component analysis or horizontal line detection, text lines can be effectively extracted for further processing. The purpose of line segmentation is to determine the number of character lines in the image and to identify the areas of these lines. This is done to exclude unnecessary components from the subsequent processes [15].

4) Character Segmentation (Bounding Box)
Once the text lines have been successfully segmented, the next step is to separate each character within those lines. This process typically begins with the detection of spaces between words, which appear as wider horizontal gaps between clusters of black pixels. Image processing algorithms then place bounding boxes around each detected character. A bounding box is a rectangle that surrounds each character, identifying the top, bottom, left, and right boundary coordinates of the word. With the use of bounding boxes, each character can be extracted as a separate entity. The following bounding box process marks line blocks with boxes. These markers allow you to see individual objects. The following explains the dilation processing test stage:
   a) The bounding box process marks line blocks in the dilated image.
   b) Line blocks become individual objects in the dilated image.

   The bounding process separates one object from another and computes features to identify each object. A bounding box is usually a rectangle determined by the coordinates of the top-left corner (x_min, y_min) and the bottom-right corner (x_max, y_max), or by center, width, and height. In the dilated image, the bounding process is used to separate line block objects. This process performs object segmentation to separate one object from another based on pixel connectivity in the dilated image. Subsequently, the bounding process computes object features used to label objects that are separated from the line block. Objects are marked with a red box using the computed height and width dimensions from the labeling process [16].

5) Dilation
Dilation is a process aimed at thickening the white pixels of the object to be detected, making it easier for the computer to detect the object [17]. Dilation is one of the morphological image processing methods related to the shape structure of objects. Specifically, dilation is the process of adding pixels to the boundaries of objects in a digital image. In other words, dilation adds pixels to the object's boundaries, so after the dilation operation, the size of the object in the image increases [18].

6) Morphological Operations
Morphological image processing refers to important techniques in image processing that alter the shape and structure of objects in the original image [19]. To make shapes (structures) more recognizable, morphological operations are used. Morphological image processing is usually performed by applying a structuring element to the image in a manner similar to convolution. The structuring element (SE) is a critical component for morphological operations [20].

7) Python
Python is a high-level programming language that can execute a variety of instructions directly (interpreted), using object-oriented programming and dynamic semantics to provide readable syntax [21]. Python offers many libraries such as OpenCV, Pillow, NumPy, and Matplotlib, which facilitate image segmentation processes, including object detection and annotation using bounding boxes.

8) Google Colaboratory
Google Colaboratory, also known as Google Colab, is a free tool for research purposes that utilizes cloud storage. Google Colaboratory functions similarly to Jupyter Notebook, but it is accessible online and free of charge [22]. Google Colaboratory allows users to run code with GPU resources without requiring installation on a local computer. This greatly accelerates the processing of large datasets. Additionally, its direct integration with Google Drive simplifies file management, such as datasets and annotated images during the segmentation process.

4. Result and Discussion

4.1 Results

4.1.1 Data Collection

The initial stage of this research involved collecting data from script artifacts located in the OKU Timur region. The collected data was then processed by researchers who are part of the ISRG research team under the research title "Development of a Mobile-Based Transliteration Application for Ulu Script Variants in OKU Timur." In this study, questionnaires were carefully designed to cover various character variations of the OKU Timur script. Each questionnaire consists of 225 OKU Timur script characters, which were then distributed to respondents for completion. Subsequently, the collected OKU Timur script characters were processed and converted into questionnaires. These questionnaires were later used for the Image Preprocessing stage. Below is an example of a questionnaire page that has been completed by respondents and processed at the Image Preprocessing stage.

Figure 3. Questionnaire Page

4.1.2 Line Segmentation

1) Displaying OKU Timur Script Images

```
# Fungsi untuk memeriksa apakah file berada di Google Drive
def is_from_drive(file_path):
    return '/content/drive/' in file_path

# Path ke gambar
image_path = '/content/drive/MyDrive/dataset fiks - Copy/3/03.jpg'

# Cek apakah file berasal dari Google Drive
if not is_from_drive(image_path):
    raise ValueError("Error: Gambar harus berasal dari Google Drive!")

# Membaca dan mengonversi gambar
img = cv2.imread(image_path)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

# Menyesuaikan kontras dan kecerahan
alpha = 1.0  # Kontras (1.0 - 3.0)
beta = 15    # Kecerahan (0-100)

img = cv2.convertScaleAbs(img, alpha=alpha, beta=beta)

# Menampilkan gambar
plt.imshow(img)
plt.axis('off')
plt.show()
```

Figure 4. Code for Displaying Images

The images processed in this step were obtained from the previous stage, namely the Image Preprocessing stage, and uploaded to Google Drive for segmentation processing. The code above is designed to process and display images stored in Google Drive by first verifying the source of the file. The initial step in this code is to check whether the image file is located in Google Drive by searching for the string '/content/drive/' in the file path. If the image is not from that location, the code will halt the process and provide an error message. Once verification is successful, the image is read using OpenCV, which by default reads images in the BGR (Blue-Green-Red) color format. To ensure compatibility with Matplotlib, the image is then converted to the RGB (Red-Green-Blue) color format, which is more commonly used for visualization. The next step in this code is to adjust the image's contrast and brightness. These adjustments are made using the variable alpha for contrast and beta for brightness. The alpha value controls the contrast level, where a value of 1.0 means the contrast remains unchanged, while higher values will increase the image's contrast. Meanwhile, beta controls the brightness, with higher values making the image appear brighter. These adjustments are applied to the image using the cv2.convertScaleAbs() function. After the contrast and brightness have been adjusted, the image is displayed using Matplotlib with the axes hidden to focus on the image. Thus, this code allows users to read, verify, adjust, and display images from Google Drive with easy control over contrast and brightness.

2) Dilation for Line Segmentation

```
# Dilasi untuk segmentasi baris
kernel = np.ones((30,70), np.uint8)
dilated = cv2.dilate(binary, kernel, iterations = 1)
plt.imshow(dilated, cmap='gray')
```

Figure 5. Code for Dilation in Line Segmentation

After successfully displaying the image, it is processed in the dilation stage. This code is used to perform dilation on a binary image, which is a technique in image processing to expand the area of bright objects in the image. First, a kernel or structuring element with a size of 30x70 pixels is created using np.ones, where each element has a value of 1. This kernel acts as a "stamp" used in the dilation process. The cv2.dilate function is then applied to the binary image (binary) with the kernel, and the dilation process is performed once (as specified by iterations = 1). This dilation causes the bright objects in the image to become thicker or larger, which is useful for connecting separated elements, such as text lines in document segmentation.

3) Finding Contours in the Image

```
# Temukan kontur dalam gambar tersegmentasi
(contours, hierarchy) = cv2.findContours(dilated.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)
sorted_contours_lines = sorted(contours, key = lambda ctr : cv2.boundingRect(ctr)[1]) # (x, y, w, h)
```

Figure 6. Code for Finding Contours

The above code is used to find and sort contours in an image that has undergone dilation. First, the cv2.findContours function is used to detect contours in the segmented image (which has previously undergone dilation). This function returns two values: contours, which contains a list of all detected contours, and hierarchy, which contains information about the hierarchical relationship between contours. The parameter cv2.RETR_EXTERNAL ensures that only the outermost contours are retrieved, while cv2.CHAIN_APPROX_NONE stores all points in the contour without reduction. Once the contours are found, this code sorts the contours based on their vertical position (the y coordinate) using the sorted function. This sorting process is important in applications such as text line segmentation, where the order of contours determines the order of the lines. The function cv2.boundingRect(ctr) is used to obtain the coordinates and size of the bounding box for each contour, and key = lambda ctr: cv2.boundingRect(ctr) [1], ensures that sorting is done based on the y value, which is the vertical position of the contour in the image.

4) Saving and Displaying the Result of Line Segmentation

```
# Gambar bounding box pada kontur
for ctr in sorted_contours_lines:
    x, y, w, h = cv2.boundingRect(ctr)
    cv2.rectangle(img2, (x,  y), (x + w, y + h), (40, 100, 250), 2)

# Menyimpan gambar hasil segmentasi baris
output_path_lines = 'segmentasi_baris.jpg'
if not cv2.imwrite(output_path_lines, img2):
    raise IOError(f"Failed to save image {output_path_lines}")

# Menampilkan gambar dengan bounding box
plt.imshow(cv2.cvtColor(img2, cv2.COLOR_BGR2RGB))
plt.title('Segmentasi Garis dengan Bounding Box')
plt.show()
```

Figure 7. Code for Saving and Displaying the Result Image

This code functions to draw bounding boxes around each previously sorted contour and then save and display the result image. First, the code iterates through each contour that has been sorted by its vertical position (sorted_contours_lines). For each contour, the top-left corner coordinates (x, y), width (w), and height (h) of the bounding box are calculated using cv2.boundingRect(ctr). The bounding box is then drawn on the original image (img2) using cv2.rectangle, with the box color specified by the RGB value (40, 100, 250) and a line thickness of 2 pixels. After all bounding boxes are drawn, the resulting image is saved to a file named segmentasi_baris.jpg using cv2.imwrite. If the saving process fails, the program will output an error message. Finally, the image with the bounding boxes is displayed using Matplotlib (plt.imshow), and the image is converted from the BGR color format (used by OpenCV) to RGB to ensure correct display. Below is the result image from the line segmentation stage.

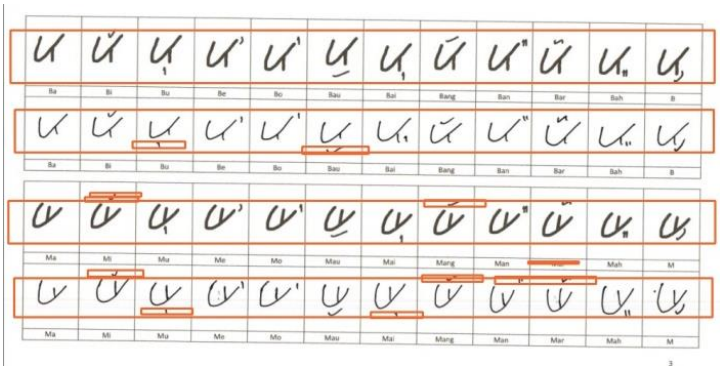

Figure 8. Result Image of Line Segmentation Stage

4.1.3 Character Segmentation (Bounding Box)

1) Dilation for Character Segmentation

```
# Dilasi untuk karakter
kernel = np.ones((8,8), np.uint8)
dilated2 = cv2.dilate(binary, kernel, iterations = 1)
plt.imshow(dilated2, cmap='gray')
```

Figure 9. Code for Dilation in Character Segmentation

After the line segmentation stage has been successfully processed, the next step is dilation for the script characters in the image. This code performs dilation on the image to expand the character areas. First, a kernel with a size of 8x8 pixels is created using np.ones, where each kernel element has a value of 1. This kernel is used for the dilation process, where the cv2.dilate function will enlarge the white areas (bright objects) in the binary image (binary). Dilation is performed once, as specified by iterations = 1. The result of this dilation is an image with thicker characters and more prominent white areas, which can help connect separated parts of characters or improve the visibility of characters in the image.

2) Function to Merge Two Bounding Boxes

```
# Fungsi untuk menggabungkan dua bounding box
def merge_boxes(box1, box2):
    x1 = min(box1[0], box2[0])
    y1 = min(box1[1], box2[1])
    x2 = max(box1[0] + box1[2], box2[0] + box2[2])
    y2 = max(box1[1] + box1[3], box2[1] + box2[3])
    return (x1, y1, x2 - x1, y2 - y1)
```

Figure 10. Code for Merging Two Bounding Boxes

This code defines a function called merge_boxes that is used to merge two bounding boxes into a larger box that encompasses both original boxes. This function takes two bounding boxes as input, each represented by a tuple (x, y, w, h), where x and y are the top-left corner coordinates, and w and h are the width and height of the box. To merge two bounding boxes, the function first determines the top-left corner coordinates of the merged box by taking the smallest x and y values from both boxes. Next, it determines the bottom-right corner coordinates of the merged box by taking the largest values of x + w and y + h from both boxes. Finally, the function returns the merged bounding box in the form (x1, y1, width, height), where x1 and y1 are the top-left corner coordinates, and width and height are the width and height of the merged box.

4.1.3 Morphological Operation Code

```
# Gunakan operasi morfologi untuk menggabungkan kontur kecil dengan yang lebih besar
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (8, 8))
morphed = cv2.morphologyEx(binary, cv2.MORPH_CLOSE, kernel, iterations=3)

# Temukan kontur dalam gambar biner yang telah dimorfologi
contours, _ = cv2.findContours(morphed, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
```

Figure 11. Morphological Operation Code

This code uses morphological operations to merge small contours with larger ones in a binary image. First, a square kernel with a size of 8x8 pixels is created using cv2.getStructuringElement. This kernel is used in the morphological operation with the cv2.MORPH_CLOSE method, which merges adjacent white areas (bright objects) by closing small gaps between them. This process is repeated three times (iterations=3) to ensure that small contours near larger contours will be merged, resulting in larger contours. After the morphological operation is complete, the contours present in the modified image (morphed) are found using cv2.findContours. The cv2.RETR_EXTERNAL function ensures that only the outer contours are found, and cv2.CHAIN_APPROX_SIMPLE is used to simplify the contour shapes by reducing the number of points forming the contour. The result is a list of larger contours that are combinations of previously separated small contours.

4.1.4 Filtering Contours in the Image

```
# Filter kontur kecil
min_contour_area = 2  # Batas minimum area kontur
filtered_contours = [cnt for cnt in contours if cv2.contourArea(cnt) > min_contour_area]

# Mendapatkan bounding box untuk kontur yang telah difilter
bounding_boxes = [cv2.boundingRect(contour) for contour in filtered_contours]
```

Figure 12. Code for Filtering Contours in the Image

This code functions to filter out small contours in the image, so that only larger contours are retained. First, the minimum contour area threshold is set with min_contour_area = 2, which means only contours with an area greater than 2 pixels will be retained. Then, contour filtering is performed by creating a new list,

filtered_contours, which only contains contours with an area greater than the minimum value. This filtering uses list comprehension, where cv2.contourArea(cnt) is used to calculate the area of each contour. After the smaller contours are removed, the bounding boxes for each remaining contour are calculated and stored in the bounding_boxes list. These bounding boxes are calculated using cv2.boundingRect(contour), which provides the top-left corner coordinates, width, and height of the bounding box surrounding each filtered contour. As a result, this code prepares the data for the next stage, where only relevant and sufficiently large contours will be further processed.

4.1.5 Determining the Distance for Merging Bounding Boxes and Adding Margin

```
# Jarak maksimum untuk menggabungkan bounding box
max_distance = 50
merged_boxes = []

while bounding_boxes:
    box = bounding_boxes.pop(0)
    to_merge = [box]

    for other_box in bounding_boxes[:]:
        if (abs(box[0] - other_box[0]) <= max_distance and abs(box[1] - other_box[1]) <= max_distance):
            to_merge.append(other_box)
            bounding_boxes.remove(other_box)

    if len(to_merge) > 1:
        merged_box = to_merge[0]
        for other_box in to_merge[1:]:
            merged_box = merge_boxes(merged_box, other_box)
        merged_boxes.append(merged_box)
    else:
        merged_boxes.append(box)

# Margin untuk bounding boxes
margin = 15
```

Figure 13. Code for Determining Distance and Margin

This code is used to merge bounding boxes that are close to each other into a larger box. First, the maximum allowable distance for merging bounding boxes is set with max_distance = 50. The code then processes each bounding box one by one using a while loop. For each bounding box, the code searches for other bounding boxes whose distance from the current bounding box does not exceed max_distance either horizontally or vertically. All bounding boxes that meet this distance criterion are collected in the to_merge list, and merged bounding boxes are removed from the bounding_boxes list. If there is more than one bounding box in the to_merge list, they are merged into a large bounding box using the merge_boxes function, and the result is stored in merged_boxes. If there is only one bounding box in to_merge, it is directly added to merged_boxes. Next, the code sets an additional margin with a value of margin = 15, which can be used to provide extra space around the merged bounding boxes. This ensures that the merged box covers a sufficiently wide area around it.

4.1.6 Saving and Displaying the Result Image

```
# Gambar bounding boxes pada gambar asli
for box in merged_boxes:
    x, y, w, h = box
    cv2.rectangle(img, (x - margin, y - margin), (x + w + margin, y + h + margin), (0, 255, 0), 2)

# Menyimpan dan menampilkan gambar akhir dengan bounding box
output_path = 'segmentasi_karakter.jpg'
if not cv2.imwrite(output_path, img):
    raise IOError(f"Failed to save image {output_path}")

# Menampilkan gambar akhir dengan bounding box
plt.imshow(cv2.cvtColor(img, cv2.COLOR_BGR2RGB))
plt.title('Hasil Deteksi Karakter Aksara OKU Timur dengan Bounding Box')
plt.show()
```

Figure 14. Code for Saving and Displaying the Result Image

This code is used to draw the merged bounding boxes on the original image, then save and display the results. First, the code draws each bounding box in the merged_boxes list on the original image (img). For each bounding box, the coordinates and size (x, y, w, h) are taken, and a bounding box is drawn around it by adding a margin on each side. This margin is added with a value of margin = 15 to ensure that the bounding box covers the area around the merged bounding boxes. After all bounding boxes are drawn, the image with the bounding boxes is saved to a file named segmentasi_karakter.jpg using cv2.imwrite. If the image saving fails, the code will output an error message. Finally, the detected image with bounding boxes is displayed using Matplotlib. The displayed image is converted from the BGR color format to RGB to ensure correct display,

and is given the title "OKU Timur Script Character Detection Result with Bounding Box" to provide context to the visual result. Below is the result image from the character segmentation stage.

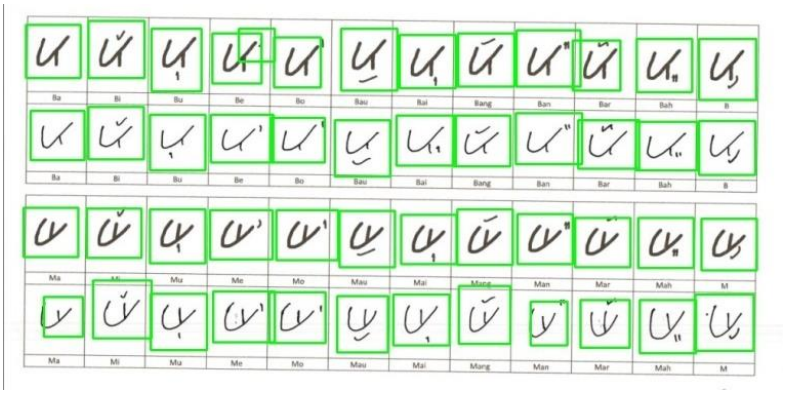

Figure 15. Result Image of Character Segmentation Stage

After the image is successfully saved, the next steps are cropping and grouping (clustering), and then the created dataset will be trained using a deep learning model so that it can later be used for image classification.

4.2 Discussion

Budiman *et al*. (2023) emphasize the importance of diverse and comprehensive data in building a robust handwritten character recognition system, especially for local scripts such as those found in OKU Timur. In this study, data collection was carried out meticulously through the distribution of questionnaires containing various character variants, ensuring that the resulting dataset truly represents the diversity of existing scripts [3]. Adipranata *et al*. (2014) highlight that the use of morphological techniques like dilation is very helpful in the image segmentation process, particularly for clarifying and connecting separated parts of characters due to scanning artifacts or irregular handwriting. In this research, dilation was applied to thicken the script components, making it easier to detect lines and separate characters [2].

Mathew *et al*. (2015) state that sorting contours based on their vertical positions is highly effective for extracting lines and characters from handwritten images. This approach was also implemented in the present study, where detected contours were sorted to ensure that each character could be isolated properly. The use of bounding boxes to segment each character further simplifies subsequent processes, especially when preparing data for machine learning models [8]. Hamanrora *et al*. (2024) also demonstrate that the bounding box technique is highly beneficial for segmenting Komering script, which shares similar characteristics with the OKU Timur script. With this approach, each character is clearly separated, making the labeling and training processes for the model much more efficient [19]. Memon *et al*. (2020) add that filtering out small, irrelevant contours is crucial to prevent the character recognition system from being disrupted by noise or artifacts from the scanned images. In this study, only contours above a certain size threshold were processed further, ensuring that the quality of the data used for model training remained high [9]. Sapitri *et al*. (2023) underline that the quality of segmented data significantly affects the performance of deep learning models. Therefore, attention to detail in every stage of segmentation and preprocessing is essential to ensure that the model can recognize characters with high accuracy [12]. This research combines scientifically proven methods and adapts them to the unique characteristics of the local script. From comprehensive data collection, careful segmentation, to well-prepared datasets for deep learning, every process was conducted systematically and thoughtfully. This forms a strong foundation for the future development of regional script recognition technology.

5. Conclusion and Recommendations

Based on the results of this study, a total of 1,020 images were processed. These images were obtained from questionnaires that had been completed and then scanned using a scanner. Subsequently, the images underwent a preprocessing stage aimed at improving their quality before entering the segmentation phase. This research utilized the Bounding Box method to process images of the OKU Timur script that had passed through the preprocessing stage. This method effectively helped to separate the characters on the respondents' answer sheets. By using the Bounding Box method, the segmented character images could be easily processed in the following stages, namely cropping and clustering. A suggestion for improving the segmentation process is to develop or add code that enables the input of multiple images at once. Currently,

images must be input into the program one by one, which can be very time-consuming if there are many images to process.

References

[1] Setiyawan, P. A., & Bayupati, I. P. A. (2014). Balinese Alphabet Sebagai Aplikasi Media Pembelajaran Aksara Bali Berbasis Android Mobile Platform. *Jurnal Merpati*, 2(2), 226–237.

[2] Adipranata, R., Siwalankerto, J., & Telp, S. (2014). Kombinasi metode morphological gradient dan transformasi watershed pada proses segmentasi citra digital. *Jurnal Informatika Petra*, (031).

[3] Budiman, A., Fadlil, A., & Umar, R. (2023). Improving the results of learning nglegena Javanese handwriting using backpropagation artificial neural network. *Edunesia Jurnal Ilmiah Pendidikan*, 4(1), 259–269. https://doi.org/10.51276/edu.v4i1.339

[4] Darma, I. W. A. S., & Ariasih, N. K. (2018). Handwritten Balinesse Character Recognition using K-Nearest Neighbor. *International Association for Convergence Science & Technology*, 119-123. https://doi.org/10.31227/osf.io/z6m8u

[5] Darma, I. W. A. S. (2019). Implementation of Zoning and K-Nearest Neighbor in Character Recognition of Wrésastra Script. *Lontar Komputer: Jurnal Ilmiah Teknologi Informasi*, *10*(1), 9. https://doi.org/10.24843/lkjiti.2019.v10.i01.p02

[6] Kantorov, V., Oquab, M., Cho, M., & Laptev, I. (2016, September). Contextlocnet: Context-aware deep network models for weakly supervised localization. In *European conference on computer vision* (pp. 350-365). Cham: Springer International Publishing. https://doi.org/10.1007/978-3-319-46454-1_22

[7] Kaur, S., & Sagar, B. B. (2023). Efficient Scalable Template-Matching Technique for Ancient Brahmi Script Image. *Computers, Materials & Continua*, *75*(1), 1541–1559. https://doi.org/10.32604/cmc.2023.032857

[8] Mathew, C. J., Shinde, R. C., & Patil, C. Y. (2015, March). Segmentation techniques for handwritten script recognition system. In *2015 International Conference on Circuits, Power and Computing Technologies [ICCPCT-2015]* (pp. 1-7). IEEE. https://doi.org/10.1109/iccpct.2015.7159397

[9] Memon, J., Sami, M., Khan, R. A., & Uddin, M. (2020). Handwritten optical character recognition (OCR): A comprehensive systematic literature review (SLR). *IEEE access*, *8*, 142642-142668. https://doi.org/10.1109/access.2020.3012542

[10] Papadopoulos, D. P., Uijlings, J. R., Keller, F., & Ferrari, V. (2016). We don't need no bounding-boxes: Training object class detectors using only human verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 854-863). https://doi.org/10.1109/cvpr.2016.99

[11] Rasyidi, M. A., Bariyah, T., Riskajaya, Y. I., & Septyani, A. D. (2021). Classification of handwritten Javanese script using random forest algorithm. *Bulletin of Electrical Engineering and Informatics*, *10*(3), 1308-1315. https://doi.org/10.11591/eei.v10i3.3036

[12] Sapitri, W., Kunang, Y. N., Yadi, I. Z., & Mahmud, M. (2023). The Impact of Data Augmentation Techniques on the Recognition of Script Images in Deep Learning Models. *Jurnal Online Informatika*, *8*(2), 169-176. https://doi.org/10.15575/join.v8i2.1073

[13] Gumiwang, Z. Y. M., Zahhar, A. H. N., & Maulana, H. (2023). Perbandingan Segmentasi Citra Menggunakan Algoritma K-Means Dan Algoritma Fuzzy C-Means. *Jurnal Manajamen Informatika Jayakarta*, *3*(1), 21-26. http://journal.stmikjayakarta.ac.id/index.php/JMIJayakarta

[14] Sari, N. L. K., Hartoyo, P., & Ajrun, A. (2022). Analisis Karakteristik Segmen Pada Citra Mamografi Dengan Menggunakan Metode Segmentasi Watershed. *JURNAL PEMBELAJARAN FISIKA*, *11*(2), 59-64. https://doi.org/10.19184/jpf.v11i2.31643

[15] Septiarini, A. (2016). Segmentasi Karakter Menggunakan Profil Proyeksi. *Informatika Mulawarman: Jurnal Ilmiah Ilmu Komputer*, *7*(2), 66-69.

[16] Maedjaja, F. (2021). Sistem Deteksi Teks pada Cover Buku dengan Pendekatan Karakter Teks. *Infact: International Journal of Computers*, *6*(2).

[17] Riandini, R., & Kuncoro, D. (2023). Estimasi Panjang Antrean Kendaraan pada Persimpangan Jalan Raya dengan Sensor Kamera Menggunakan Metode Queue Length Estimation. *Journal of Computer Engineering, Network, and Intelligent Multimedia*, *1*(1), 14-20. https://doi.org/10.59378/jcenim.v1i1.4

[18] Sutama, V. M., Magdalena, R., & Wijayanto, I. (2018). Identifikasi Objek Dominan Citra Digital Menggunakan Metode Markov Random Field (mrf). *eProceedings of Engineering*, *5*(3), 4859–4865. https://openlibrarypublications.telkomuniversity.ac.id/index.php/engineering/article/view/7839

[19] Hamanrora, M. D., Kunang, Y. N., Yadi, I. Z., & Mahmud. (2024). Image segmentation of Komering script using bounding box. *Indonesian Journal of Electrical Engineering and Computer Science*, 35(3), 1565–1578. https://doi.org/10.11591/ijeecs.v35.i3.pp1565-1578

[20] Nainggolan, I. B., Magdalena, R., & Fuadah, R. Y. N. (2018). Matched Filter Dan Operasi Morfologi Untuk Estimasi Derajat Kebengkokan Tulang. *eProceedings of Engineering*, *5*(3), 5108.

[21] Rahmadhika, M. K., & Thantawi, A. M. (2021). Rancang Bangun Aplikasi Face Recognition Pada Pendekatan CRM Menggunakan Opencv Dan Algoritma Haarcascade. *IKRA-ITH INFORMATIKA: Jurnal Komputer Dan Informatika*, *5*(1), 109-118.

[22] Maesaroh, M., Padilah, T. N., & Jaman, J. H. (2023). Penerapan Algoritma K-Means Clustering Pada Pengelompokan Daerah Penyebararan Diare Di Provinsi Jawa Barat. *JATI (Jurnal Mahasiswa Teknik Informatika)*, *7*(4), 2783-2787. https://doi.org/10.36040/jati.v7i4.7208.