Journal of Computer Science

Original Research Paper

An Approach to Recognise Odia Characters by Extracting Features from Fragmented Quadrants

Aradhana Kar and Sateesh Kumar Pradhan

Department of Computer Science and Applications, Utkal University, Bhubaneswar, Odisha, India

Article history
Received: 18-10-2022
Revised: 09-11-2022
Accepted: 15-11-2022

Corresponding Authors:
Aradhana Kar
Department of Computer Science and Applications,
Utkal University,
Bhubaneswar, Odisha, India
Email: aradhana140587@gmail.com

Abstract: The system introduced in this study concentrates on the recognition of Odia characters. The input to the system is an image in .jpg format which has been created by scanning a document or converting a document to an image by software. The document consists of all Odia alphabets written using the font family 'AkrutiOriAshok-99' in a particular font size in bold style. The font sizes that have been considered are 18, 20, 22, 24, 26, 28, 36, 48, and 72. The noise is removed and line texts are segmented from the input image. Then individual alphabets are segmented from a line text using the Character_Segment part of the system proposed in this study. Then the features are extracted from the segmented individual alphabet and the features are searched in the dictionary of features stored in an excel file for a correct match one by one. The proposed system is also responsible for creating the dictionary of features of all Odia alphabets. Hence, the system in this study has four parts, Segment_Line, Character_Segment, Build_Dictionary, and Recognise_Character. According to the testing results, the system proposed in this study has achieved a correctness accuracy of recognition of Odia alphabets of 99.8%.

Keywords: Recognise Odia Characters, Extracting Features, Odia Characters, Fragmented Quadrants

Introduction

Recognition of characters in an image by following several steps of an algorithm is known as Optical Character Recognition (OCR). Old scripts can be damaged as a result the knowledge in those scripts can get lost. Digitization of the scripts and recognition of text in the old scripts save the knowledge in it as well as convert the digitized script in an editable form. Nowadays, hard copies of documents are frequently scanned and posted in many digital forums. OCR can convert these scanned documents into an editable form. This study concentrates on the recognition of the Odia characters. Odia language consists of three categories of the alphabet: *"Swara Barna"*, *"Byanjana Barna"* and *"Atirikta Barna"*. All the alphabets in these categories sum up to 48 alphabets (excluding 0, Θ and 8) (Rao, 1901) (Fig. 1). When *"Swara Barna"* is used with *"Byanjana Barna"* or *"Atirikta Barna"*, the *"Swara Barna"* becomes a symbol and gets associated with *"Byanjana Barna"* and *"Atirikta Barna"*. This symbol is known as *"Matra"* (Mohapatra and Das, 1952). Only 0, Θ, and 8 are used with all alphabets of *"Swara Barna"*, the remaining alphabets of *"Byanjana Barna"* and all alphabets of *"Atirikta Barna"*.

This study concentrates on a system that takes an image in .jpg format which has been created by scanning a document or converting a document to an image by software. The document consists of all Odia alphabets written using the font family 'AkrutiOriAshok-99' in a particular font size in bold style. The font sizes that are been considered are 18, 20, 22, 24, 26, 28, 36, 48, and 72. The process of extraction of features from a character and finding a correct match follows all the sub-modules of the find match module of the system (Kar and Pradhan, 2022).

Unlike the system in (Kar and Pradhan, 2022) where the image was divided into four quadrants, the proposed system divides the images of the Odia alphabets stored in a directory named *'Dictionary'* and the input image is divided into sixteen quadrants. As per the research, the system in (Kar and Pradhan, 2022) is not able to trace a lot of features in a quadrant. Hence, it has been decided to break the image into sixteen quadrants so that more features can be extracted from an image which will result in greater accuracy.

DOI: 10.3844/jcssp.2023.57.74

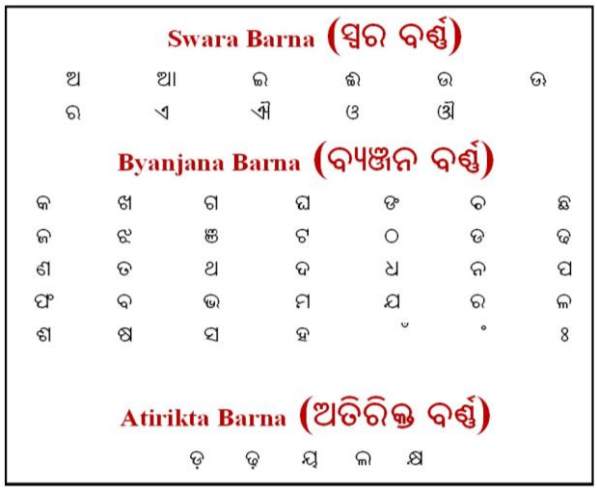

Fig. 1: Alphabets of Odia language

This study consists of four parts: Build_Dictionary, Segment_Line, Character_Segment, and Recognise_Character. The Build_Dictionary part has the following modules: Pre-Processing, Find_Direction, Get_Pattern_Left, Get_Pattern_Right, Sub_Quad_Pattern, Remaining_Part, Write_Excel, Find_Common_Feature, and Group_Vacant_Quads. The modules Get_Pattern_Left, Get_Pattern_Right, Sub_Quad_Pattern, Remaining_Part, and Write_Excel follow the steps of the modules GettingFeaturesLeft, Getting_Features_Right, Visit Sub Quad, Remaining Sub_Quad and Write_To_Excel of Feature Extraction in Dictionary_Building of the system in (Kar and Pradhan, 2022) respectively. The Find_Common_Feature module explained in this system, finds common features of each Odia alphabet by applying Longest Common Subsequence (LCS) on the features extracted from the Odia alphabet in all font sizes repeatedly. The Find_Common_Feature consists of two sub-modules; Create_LCS and Find_LCS. The Group_Vacant_Quads module explained in this system groups Odia alphabets based on the first vacant quadrant found. When an Odia alphabet is divided into sixteen quadrants, it may happen that some quadrants do not contain any part of the alphabet. These quadrants are known as Vacant Quadrants. An Odia alphabet may or may not have at least one vacant quadrant. The work of the Group_Vacant_Quads is to iterate through the quadrants of an alphabet and group it based on the first vacant quadrant found. The Segment_Line part segments line texts from the document image using the Line Segmenting module of the system described in the research paper (Kar and Pradhan, 2021). The Character_Segment part has been explained in this study and this part segments each character from the segmented line texts. The Recognise_Character part consists of modules: Pre-Processing, Find_Direction, Get_Pattern_Left, Get_Pattern_Right, Sub_Quad_Pattern, and Remaining_Part for extraction of features and these features are written to the excel file, '*InputFile16.xlsx'* using the module Write_Excel. The Search_For_Match module has been used to find the correct match of the input image. The system proposed in this study has achieved correctness accuracy in the recognition of Odia alphabets of 99.8%.

Python programming language has been used for implementing the system. List and Dictionary concepts of python have been used in the proposed system. List behaves as a dynamic array in python in which data can be easily appended and removed (Solem, 2012; Shaw, 2013; Lee, 2014; Beazley and Jones, 2011; Downey, 2019). Dictionary is an unordered collection of data where items are stored in a particular index. Unlike a list, the indices of the dictionary can be defined by the programmer (Solem, 2012; Shaw, 2013; Lee, 2014; Beazley and Jones, 2011; Downey, 2019). Files of a directory have been accessed using the so package of python (Summerfield, 2009). Finding the ceiling and usage of arrays have been done with the help of the NumPy package of python (Idris, 2011; 2012). The image processing operations such as reading an image, converting an image to a gray or binary image, and resizing images have been done using the cv2 package of python (Howse *et al*., 2016). The binary image is a two-tone image where the value of pixels can either be 0 or 1 (Otsu, 1979; Chityala and Pudipeddi, 2020; Howse *et al*., 2016; Nixon and Aguado, 2019). A Gray-scale image is an image where the value of intensities of pixels ranges from 0 to 255 (Sridhar, 2013; Gonzalez and Woods, 2017). The images have been generated for display or saved in a directory by using the matplotlib package of python (Johansson, 2019; Poladi, 2018). The extracted features have been written to the excel file using the openpyxl package of python (Gazoni and Clark, 2022). LCS has been used to find the common features from the features extracted from the Odia alphabets in all font sizes and stored in an excel file, *'CFeature16.xlsx'*. LCS finds the longest subsequence when two strings are given as input (Cormen *et al*., 2009; Agarwal, 2008; Narasimha Karumanchi, 2017; Skiena, 2013; Vermani and Vermani, 2019). In other words, the Build_Dictionary part takes the images of Odia alphabets from the *'Dictionary'* directory and the extracted features from the images are written in an excel file named 'DictionaryFeatures16.xlsx'. On the other side, an image of a document is given as input to the proposed system. The image consists of Odia alphabets in the font family *'*AkrutiOriAshok-99*'* in a particular font size in bold style. The input goes through three parts Segment_Line, Character_Segment, and Recognise_Character to segment the lines from the document, segment characters from the lines, and recognize segmented characters respectively.

In past works, recognizing characters in any language went through steps like pre-processing, line segmentation, word segmentation, character segmentation, features extraction from the character, and finding a match for the extracted features in a dictionary. Significant work had been done both for hand-written documents and printed documents.

For feature extraction from characters, different ways had been adopted in past works. The system in (Kar and Pradhan, 2022) created a dictionary of images of Odia alphabets in different font sizes and stored them in a directory Dictionary. The features were extracted from the images present in Dictionary by following some number of modules and the extracted features were written in an excel file, DictionaryFeatures.xlsx. A common feature has been extracted for each Odia alphabet by applying Longest Common Subsequence (LCS) repeatedly on the extracted features of Odia alphabets in different font sizes and the common feature for each Odia alphabet is written in the excel file, CommonFeature.xlsx. In the system (Tripathy and Pal 2006) and in the system (Pal *et al.*, 2003), features were extracted from the characters by using the technique called water reservoir. The reservoir was obtained by the accumulation of water poured from the top or the bottom of the numerals. Top reservoirs were formed when water was poured from the top and bottom reservoirs were formed when water was poured from the top after rotating the component by 180^{o}. Water reservoirs were the white regions of the component. The features that were considered in the scheme were: The number of reservoirs, position of reservoirs concerning the bounding box of the touching pattern, shape, and size of the reservoirs, center of gravity of the reservoirs, and relative positions of the reservoirs. The system in (Das *et al*., 2017) divided the image of each character into 9 zones and in each zone found 12 strokes. Hence, the feature of a character was represented in 1 × 108 vector format. This research was carried out on Kalinga font. In the system (Padhi *et al*., 2013), features were extracted by finding the mid value of the image called the centroid. In the first phase, the image was then divided into three equal halves row-wise and two halves column-wise, making it six zones. The distance between the pixel value and the centroid was calculated and this was done for all pixels for a zone and then the average distance was calculated for that zone. The angle between the image centroid and the pixel was calculated and this was done for each pixel in a zone. Then the average of the angles was calculated. In the second phase, the image was divided into nine zones and the procedure that was carried out in the first phase was also followed in the second phase. The first phase output six average distances and six average angles. The second phase also outputs nine standard deviations, nine average distances, and nine average angles as features. The system in (Goswami and Mitra, 2016) concentrated on Gujarati characters. The features in this system were extracted by finding high-level as well as low-level strokes. The high-level strokes were described in (Goswami and Mitra, 2018) and the low-level strokes were described in (Goswami and Mitra, 2016). The proposed method by the system by (Goswami and Mitra, 2016) started scanning from the center region of the character in left to right order and extracts all junction points. The 3 × 3 neighborhood of each junction point was then scanned in clockwise order to obtain the starting point of each high-level stroke. The high-level stroke ends at the endpoint or until the next junction point is not reached using the contour tracing method.

For recognition of characters based on the extracted features, the system in (Kar and Pradhan, 2022) searched the extracted features of the input image in the CommonFeature.xlsx excel file. The accuracy achieved in the recognition of the character was 98.1%. In the system (Tripathy and Pal 2006) and in the system (Pal *et al.*, 2003), the recognition was done based on the features that consist of several reservoirs, the position of reservoirs concerning the bounding box of the touching pattern, shape, and size of the reservoirs, center of gravity of the reservoirs and relative positions of the reservoirs. The accuracy achieved in the character recognition by the system (Tripathy and Pal 2006) and the system (Pal *et al.*, 2003) was 95.1% and 94.8% respectively. The system proposed by Das *et al*. (2017) used Structural Similarity Index as it is based on the concept that the structure of the image is independent of the illumination. The accuracy achieved in character recognition was 92%. The system in (Padhi and Senapati, 2013) used Artificial Neural Network for classification. The system in (Goswami and Mitra, 2018) used a finite state machine to identify a high-level stroke. For classification, the system used Naive Bayes Classifier and Hidden Markov Model. The overall accuracy achieved using the Naive Bayes Classifier and Hidden Markov Model in recognizing characters were 93.26 and 96.87% respectively.

To segment lines, the system in (Tripathy and Pal, 2006) divided the document to find vertical stripes. For segmenting words from lines, the system in (Tripathy and Pal, 2006) divided the document to find vertical stripes. Based on the vertical projection profile and structural features of Odia characters, text lines were segmented into words.

Materials and Methods

System Architecture

The overall system consists of four parts, *'Build_Dictionary','Segment_Line','Character_Segment'*, and *'Recognise_Character'*. Build_Dictionary part is responsible for building a dictionary of features from a directory named *'Dictionary'* that contains the images of

Odia alphabets in the font family *'AkritiOriAshok-99'* in different font sizes in bold style. On the other side, a document consisting of Odia alphabets in the font family *'AkritiOriAshok-99'* in a particular font size in bold style, is given as input to the Segment_Line part that segments the line texts from the document, and the segmented line texts are stored in the directory, *'Line Segments'*. Then Character_Segment part segments the individual alphabet from the segmented line texts present in the directory, *'Line Segments'*, and the segmented alphabets are stored in the directory, *'alphabets'*. Then segmented alphabets are accessed one at a time and given as input to the 'Recognise_Character' part. The Recognise_Character part first extracts features from the input alphabet using some modules and then finds the appropriate match in the repository of common features.

Build_Dictionary

This part concentrates on creating a dictionary of extracted features by applying a feature extraction procedure on the images of Odia alphabets present in a directory **'***Dictionary'*. The document consists of Odia alphabets which are written in the font family *'*AkrutiOriAshok-99*'* in a particular font size in bold style. In this way, 9 documents consisting of 48 alphabets are created in nine different font sizes such as 18, 20, 22, 24, 26, 28, 36, 48, and 72. These documents are either scanned or converted to an image by the software. From each document, 48 images have been created and they are stored in a directory. For example, a document consisting of Odia alphabets in font size 18 is stored in a directory. In this way, nine directories are created and these nine directories are stored in the directory **'***Dictionary'*. The nine directories in the directory *'Dictionary'* are accessed one by one and for each directory, a sheet is created in an excel file, *'*DictionaryFeatures16.xlsx*'*. In Fig. 2, the image of each Odia alphabet, **'***A'* is accessed from each directory of *'Dictionary'* and it goes through the modules, Pre-Processing, Find_Direction, Get_Pattern_Left, Get_Pattern_Right, Sub_Quad_Pattern and Remaining_Part for feature extraction. The extracted feature is written in the particular row and column of a particular sheet of the excel file, **'**DictionaryFeatures16.xlsx*'* using the module, Write_Excel. Then, the excel file, *'*DictionaryFeatures16.xlsx*'* is given as input to the Find_Common_Feature to find the common feature of an alphabet and this common feature is written in the excel file, *'CFeature16.xlsx'*. Then, the Group_Vacant_Quads module takes *'CFeature16.xlsx'* as input and groups the features of the alphabets based on vacant quadrants, and these grouped features are written in the excel file, *'VacantQuadrants.xlsx'* in different sheets.

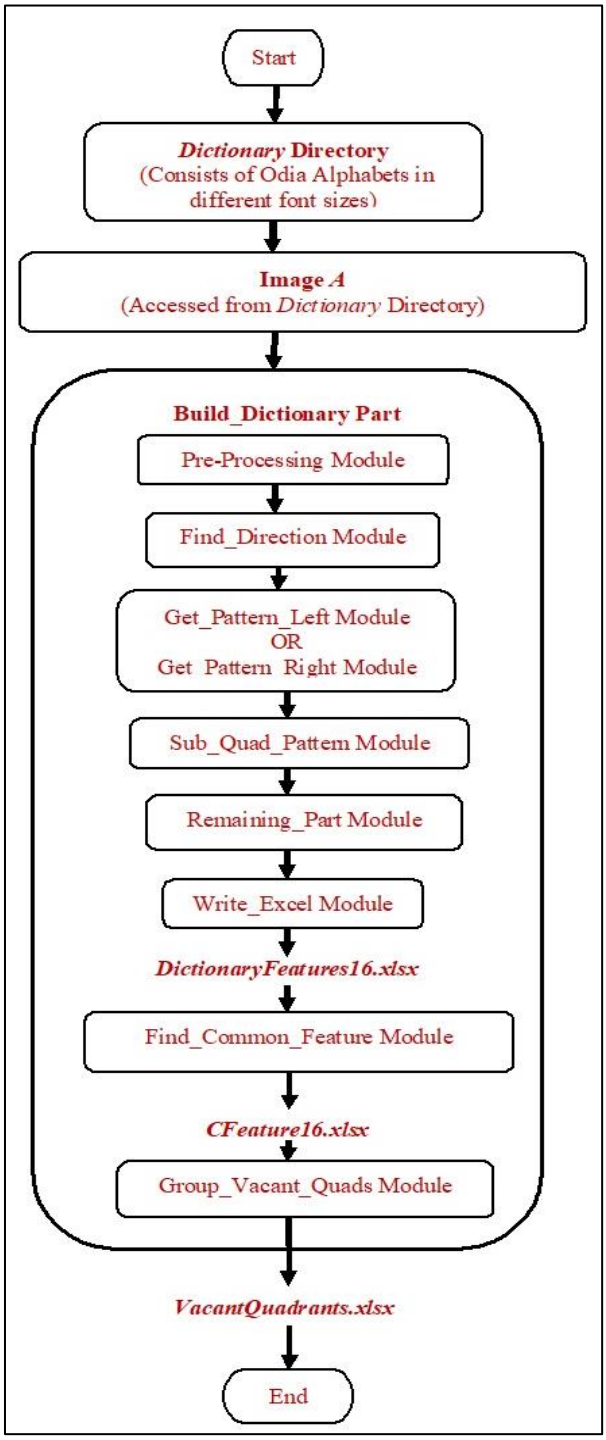

Fig. 2: Overall working of Build_Dictionary

Pre-Processing Module

The images of the Odia alphabet are accessed one by one from each directory in **'***Dictionary'*. The image '*A'* in any of the directories of *'Dictionary'* is given as input to this module.

Algorithm:

1. The image '***A'*** is converted to a gray-scale image and stored in '***B'***.
2. The white spaces at the margins that enclose the text in image '***B'*** are removed using the RemoveBoundarySpaces phase of the RemoveNoise module of (Kar and Pradhan, 2021), and the result is stored in ***'C'***.
3. The image '***C'*** has been transformed into a dimension of 64 x 64 by using inter-cubic interpolation and the result is stored in '***D'***.
4. The image '***D'*** is then transformed into a binary image and stored in '***E'.***
5. Image ***'E'*** is partitioned into four equal slices, both in the horizontal and vertical directions. In this way, image ***'E'*** is partitioned into sixteen equal quadrants. The dimension of the image ***'E'*** is row16 x col16 (row16 = 64 and col16 = 64), where ***'row16'*** is the number of rows and ***'col16'*** is the number of columns.
6. The values needed for dividing the image ***'E'*** horizontally are calculated in the following ways:

$$row1 = \left\lfloor \frac{row16}{4} \right\rfloor$$

$$row2 = 2 * row1$$

$$row3 = row2 + row1$$

7. The values needed for dividing the image ***'E'*** vertically are calculated in the following ways:

$$col1 = \left\lfloor \frac{col16}{4} \right\rfloor$$

$$col2 = 2 * col1$$

$$col3 = col2 + col1$$

8. The sixteen quadrants are found from ***'E'*** by using the values calculated in steps **6** and **7**. The sixteen quadrants are identified as ***'First'***, ***'Second'***, ***'Third'***, ***'Fourth'***, ***'Fifth'***, ***'Sixth'***, ***'Seventh'***, ***'Eighth'***, ***'Ninth'***, ***'Tenth'***, ***'Eleventh'***, ***'Twelfth'***, ***'Thirteenth'***, ***'Fourteenth'***, ***'Fifteenth'*** and ***'Sixteenth'***.
   First = E[0:row1, 0:col1]
   Second = E[0:row1, (col1 + 1):col2]
   Third = E[0:row1, (col2 + 1):col3]
   Fourth = E[0:row1, (col3 + 1):col16]
   Fifth = E[(row1 + 1):row2, 0:col1]
   Sixth = E[(row1 + 1):row2, (col1 + 1):col2]
   Seventh = E[(row1 + 1):row2, (col2 + 1):col3]
   Eighth = E[(row1 + 1):row2, (col3 + 1):col16]
   Ninth = E[(row2 + 1):row3, 0:col1]
   Tenth = E[(row2 + 1):row3, (col1 + 1):col2]
   Eleventh = E[(row2 + 1):row3, (col2 + 1):col3]
   Twelfth = E[(row2 + 1):row3, (col3 + 1):col16]
   Thirteenth = E[(row3 + 1):row16, 0:col1]
   Fourteenth = E[(row3 + 1):row16, (col1 + 1):col2]
   Fifteenth = E[(row3 + 1):row16, (col2 + 1):col3]
   Sixteenth = E[(row3 + 1):row16, (col3 + 1):col16]
9. Call ***Find_Direction(QNo, quad, DItem, DInItem, DPath, SName)*** for the quadrants ***First***, ***Second***, ***Third***, ***Fourth***, ***Fifth***, ***Sixth***, ***Seventh***, ***Eighth***, ***Ninth***, ***Tenth***, ***Eleventh***, ***Twelfth***, ***Thirteenth***, ***Fourteenth***, ***Fifteenth***, and ***Sixteenth*** where ***QNo*** is the quadrant number among the sixteen quadrants. Here, QNo = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, and 16 ***quad***s can be ***First***, ***Second***, ***Third***, ***Fourth***, ***Fifth***, ***Sixth***, ***Seventh***, ***Eighth***, ***Ninth***, ***Tenth***, ***Eleventh***, ***Twelfth***, ***Thirteenth***, ***Fourteenth***, ***Fifteenth***, and ***Sixteenth.***
   DItem is the i^{th} sub-directory in ***'Dictionary'***.
   DItem = 1, 2, 3, 4, 5, 6, 7, 8 and 9. For any value of DItem, a sheet in **'DictionaryFeatures16.xlsx'** is created with the name the same as the value of DItem. For example, if DItem = 2 then a sheet with the name ***'2'*** is created in the excel file.
   DInItem is the i^{th} image file of the $DItem^{th}$ subdirectory of ***'Dictionary'***. DInItem = 1, 2, 3,....., 48 where ***'48'*** is the total number of image files in the $DItem^{th}$ directory.
   DPath is the absolute path of the excel file, **'DictionaryFeatures16.xlsx',** in which the features of the alphabet are being written.
   SName is the file name of the image file present in any $DItem^{th}$ sub-directory of ***'Dictionary'***.
   Suppose QNo = 11, quad = eleventh, DItem = 4, DInItem = 14 then a sheet with the name ***'4'*** is created in the excel file, **'DictionaryFeatures16.xlsx'** and its path has been stored in the ***'DPath'*** parameter and then the extracted feature is written in the '14^{th}' row (as DInItem = 14) and '11^{th}' column (as QNo = 11) of the sheet. The value, that is, the file name stored in the parameter ***'SName'*** is written in the seventeenth column.

Find_Direction Module

This module is executed for each of the quadrants First, Second, Third, Fourth, Fifth, Sixth, Seventh, Eighth, Ninth, Tenth, Eleventh, Twelfth, Thirteenth, Fourteenth, Fifteenth, and Sixteenth. Each quadrant is scanned from a particular corner to find the first black pixel and when the first black pixel is found, the coordinates of that black pixel are passed to Get_Pattern_Right or Get_Pattern_Left module. Quadrants First, Second, Third, Fifth, Sixth, Seventh, Tenth, and Thirteenth are passed to Get_Pattern_Right, and quadrants Fourth,

Eighth, Ninth, Eleventh, Twelfth, Fourteenth, Fifteenth, and Sixteenth are passed to Get_Pattern_Left for the extraction of features.

Qrow = Number of rows of ***'quad'***

Qcol = Number of columns of ***'quad'***

***Algorithm*:**

Find_Direction(QNo, quad, DItem, DInItem, DPath, SName)

1. ii and jj are initialized to ***'Qrow – 1'*** and ***'0'*** respectively.
2. IF QNo = 1 OR QNo = 5 OR QNo = 10 THEN DO FROM STEP 3 TO 6
3. REPEAT FROM STEP 4 TO 6 WHILE jj < Qcol
4. REPEAT FROM STEP 5 TO 6 WHILE ii >0
5. IF quad[ii][jj] = 0 THEN DO STEP 6
6. CALL Get_Pattern_Right(ii, jj, quad, QNo, DItem, DInItem, DPath, SName)
7. ii and jj are initialized to ***'0'***.
8. IF QNo = 2 OR QNo = 3 OR QNo = 13 THEN DO FROM STEP 9 TO 12
9. REPEAT FROM STEP 10 TO 12 WHILE ii < Qrow
10. REPEAT FROM STEP 11 TO 12 WHILE jj < Qcol
11. IF quad[ii][jj] = 0 THEN DO STEP 12
12. CALL Get_Pattern_Right(ii, jj, quad, QNo, DItem, DInItem, DPath, SName)
13. ii and jj are initialized to ***'Qrow – 1'*** and ***'Qcol – 1'*** respectively.
14. IF QNo = 4 OR QNo = 8 OR QNo = 12 OR QNo = 14 OR QNo = 16 THEN DO FROM STEP 15 TO 18
15. REPEAT FROM STEP 16 TO 18 WHILE ii>0
16. REPEAT FROM STEP 17 TO 18 WHILE jj>0
17. IF quad[ii][jj] = 0 THEN DO STEP 18
18. CALL Get_Pattern_Left(ii, jj, quad, QNo, DItem, DInItem, DPath, SName)
19. ii and jj are initialized to ***'Qrow – 1'*** and ***'0'***.
20. IF QNo = 6 OR QNo = 7 THEN DO FROM STEP 21 TO 24
21. REPEAT FROM STEP 22 TO 24 WHILE ii>0
22. REPEAT FROM STEP 23 TO 24 WHILE jj < Qcol
23. IF quad[ii][jj] = 0 THEN DO STEP 24
24. CALL Get_Pattern_Right(ii, jj, quad, QNo, DItem, DInItem, DPath, SName)
25. ii and jj are initialized to ***'0'*** and ***'Qcol - 1'***.
26. IF QNo = 9 OR QNo = 11 OR QNo = 15 THEN DO FROM STEP 27 TO 30
27. REPEAT FROM STEP 28 TO 30 WHILE ii < Qrow
28. REPEAT FROM STEP 29 TO 30 WHILE jj>0
29. IF quad[ii][jj] = 0 THEN DO STEP 30
30. CALL Get_Pattern_Left(ii, jj, quad, QNo, DItem, DInItem, DPath, SName)
31. EXIT

Get_Pattern_Left Module

The format of the Get_Pattern_Left module is Get_Pattern_Left (ii, jj, quad, QNo, DItem, DInItem, DPath, SName). ***'row'*** and ***'col'*** are the number of rows and columns of ***'quad'*** respectively. This module scans in the left direction and this starts from the first black pixel that is found in the Find_Direction Module to find a continuous trace of the black pixel in ***'quad'***. There are some scenarios.

For scenario ***'ii = 0'***, three conditions are considered and they are,

```
If quad[ii + 1][jj – 1] = 0 then
        ii = ii + 1
        jj = jj – 1
        LSubQ = Sub_Quad_Pattern(ii, jj, 0, row, 0, col)
        Get_Pattern_Left(ii, jj, quad, QNo,  DItem,
        DInItem, SName)
Else If quad[ii][jj – 1] = 0 then
        ii = ii
        jj = jj – 1
        LSubQ = Sub_Quad_Pattern(ii, jj, 0, row, 0,col)
        Get_Pattern_Left(ii, jj, quad, QNo, DItem,
       DInItem, SName)
Else If quad[ii][jj – 1] = 1 Or quad[ii + 1][jj – 1] = 1
Or ii = 0 Or jj = 0 Or ii = row – 1 Or jj = col -1 then
        Remaining_Part(quad)
        Write_Excel(DItem,DInItem,QNo,LSubQ,
        DPath, SName)
        Return
```

For scenario ***'ii = row – 1'***, three conditions are considered and they are,

```
If quad[ii – 1][jj – 1] = 0 then
        ii = ii – 1
        jj = jj – 1
       LSubQ = Sub_Quad_Pattern(ii, jj, 0,row,0,col)
       Get_Pattern_Left(ii, jj, quad, QNo, DItem,
       DInItem, SName)
Else If quad[ii][jj – 1] = 0 then
        ii = ii
        jj = jj – 1
        LSubQ = Sub_Quad_Pattern(ii, jj,0,row, 0, col)
       Get_Pattern_Left(ii, jj, quad, QNo, DItem,
       DInItem, SName)
 Else If quad[ii – 1][jj – 1] = 1 Or quad[ii][jj – 1] = 1
Or ii = 0 or jj = 0 Or ii = row – 1 Or jj = col -1 then
        Remaining_Part(quad)
        Write_Excel(DItem, DInItem, QNo, LSubQ,
        DPath, SName)
        Return
```

For scenario ***'ii ≠ row – 1 and ii ≠ 0'*** four conditions are considered and they are,

```
If quad[ii – 1][jj – 1] = 0 then
        ii = ii – 1
        jj = jj – 1
        LSubQ = Sub_Quad_Pattern(ii, jj, 0, row, 0, col)
```

```
        Get_Pattern_Left(ii, jj, quad, QNo,  DItem,
        DInItem, SName)
    Else If quad[ii][jj – 1] = 0 then
        ii = ii
        jj = jj – 1
        LSubQ = Sub_Quad_Pattern(ii, jj, 0, row, 0, col)
        Get_Pattern_Left(ii, jj, quad, QNo,  DItem,
        DInItem, SName)
    Else If quad[ii + 1][jj – 1] = 0 then
        ii = ii + 1
        jj = jj – 1
        LSubQ = Sub_Quad_Pattern(ii, jj, 0, row, 0, col)
        Get_Pattern_Left(ii, jj, quad, QNo, DItem,
        DInItem, SName)
    Else If quad[ii – 1][jj – 1] = 1 Or quad[ii + 1][jj – 1]
    = 1 Or quad[ii][jj – 1] = 1 Or ii = 0 Or jj = 0 Or ii =
    row – 1 Or jj = col -1 then
        Remaining_Part(quad)
        Write_Excel(DItem, DInItem, QNo, LSubQ,
        DPath, SName)
        Return
```

Get_Pattern_Right Module

The format of the Get_Pattern_Left module is Get_Pattern_Right(i, j, quad, QNo, DItem, DInItem, DPath, SName). ***'row'*** and ***'col'*** are the number of rows and columns of ***'quad'*** respectively. This module scans in the right direction and this starts from the first black pixel that is found in the Find_Direction Module to find a continuous trace of the black pixel in ***'quad'***. There are some scenarios.

For scenario ***'ii = 0'***, three conditions are considered and they are,

```
If quad[ii +1][jj + 1] = 0 then
    ii = ii + 1
    jj = jj + 1
    LSubQ = Sub_Quad_Pattern(ii, jj, 0, row, 0, col)
    Get_Pattern_Right(ii, jj, quad, QNo, DItem,
    DInItem, SName)
Else If quad[ii][jj + 1] = 0 then
    ii = ii
    jj = jj + 1
    LSubQ = Sub_Quad_Pattern(ii, jj, 0, row, 0, col)
    Get_Pattern_Right(ii, jj, quad, QNo, DItem,
    DInItem, SName)
Else If quad[ii][jj + 1] = 1 Or quad[ii + 1][jj + 1] = 1 Or
ii = 0 Or jj = 0 Or ii = row – 1 Or jj = col -1 then
    Remaining Part (quad)
    Write_Excel(DItem, DInItem, QNo, LSubQ,
    DPath, SName)
    Return
```

For scenario ***'ii = row – 1'***, three conditions are considered and they are,

```
If quad[ii – 1][jj + 1] = 0 then
    ii = ii – 1
    jj = jj + 1
    LSubQ = Sub_Quad_Pattern(ii, jj, 0, row, 0, col)
    Get_Pattern_Right(ii, jj, quad, QNo, DItem,
    DInItem, SName)
Else If quad[ii][jj + 1] = 0 then
    ii = ii
    jj = jj + 1
    LSubQ = Sub_Quad_Pattern(ii,jj,0,row,0,col)
    Get_Pattern_Right(ii, jj, quad, QNo, DItem,
    DInItem, SName)
Else If quad[ii – 1][jj + 1] = 1 Or quad[ii][jj + 1] = 1
Or ii = 0 Or jj = 0 Or ii = row – 1 Or jj = col -1 then
    Remaining_Part(quad)
    Write_Excel(DItem, DInItem, QNo, LSubQ,
    DPath, SName)
    Return
```

For scenario ***'ii ≠ row – 1 and ii ≠ 0'***, four conditions are considered and they are,

```
If quad[ii – 1][jj + 1] = 0 then
    ii = ii – 1
    jj = jj + 1
    LSubQ = Sub_Quad_Pattern(ii, jj, 0, row, 0, col)
    Get_Pattern_Right(ii, jj, quad, QNo, DItem,
    DInItem, SName)
Else If quad[ii][jj + 1] = 0 then
    ii = ii
    jj = jj + 1
    LSubQ = Sub_Quad_Pattern(ii, jj, 0, row, 0, col)
    Get_Pattern_Right(ii, jj, quad, QNo, DItem,
    DInItem, SName)
Else If quad[ii + 1][jj + 1] = 0 then
    ii = ii + 1
    jj = jj + 1
    LSubQ = Sub_Quad_Pattern(ii, jj, 0, row, 0, col)
    Get_Pattern_Right(ii, jj, quad, QNo, DItem,
    DInItem, SName)
Else If quad[ii – 1][jj + 1] = 1 Or quad[ii][jj + 1] = 1
Or quad[ii + 1][jj + 1] = 1 Or ii = 0 Or jj = 0 Or ii =
row – 1 Or jj = col -1 then
    Remaining_Part(quad)
    Write_Excel(DItem, DInItem, QNo, LSubQ,
    DPath, SName)
    Return
```

Sub_Quad_Pattern Module

Each of the sixteen quadrants is again partitioned into four sub-quadrants identified as ***'a'***, ***'b'***, ***'c'***, and ***'d'***. The format of the procedure of this module is Sub_Quad_Pattern (i, j, sr, er, sc, ec) where, i, j, sr, er, sc and ec are the y-coordinate of the black pixel, x-coordinate of the black pixel, start row coordinate of the quadrant, end row coordinate of the quadrant, start column coordinate of the quadrant and end column coordinate of the quadrant respectively. A list data structure identified as ***'SQu'*** in which the name of the sub-

quadrant (***'a'*** or ***'b'*** or ***'c'*** or ***'d'***) holding the first black pixel that is found in the Find_Direction Module is first appended. Then the names of the sub-quadrants for each black pixel found while scanning by Get_Pattern_Left or Get_Pattern_Right module is appended. The following values are calculated that define the boundaries of the sub-quadrants:

$$mrow = \left\lfloor \frac{er - sr}{2} \right\rfloor$$

$$mcol = \left\lfloor \frac{ec - sc}{2} \right\rfloor$$

The area from the ***'(sr)th'*** row to ***'(mrow – 1)th'*** row and from the ***'(sc)th'*** column to ***'(mcol – 1)th'*** column forms the ***'a'*** sub-quadrant. If the value of ***'i'*** lies between ***'(sr)th'*** row and ***'(mrow – 1)th'*** row and the value of ***'j'*** lies between ***'(sc)th'*** column and ***'(mcol – 1)th'*** column, ***'a'*** is appended in the list ***'SQu'***.

The area from the ***'(sr)th'*** row to ***'(mrow – 1)th'*** row and from the ***'(mcol)th'*** column to ***'(sc)th'*** column forms the ***'b'*** sub-quadrant. If the value of ***'i'*** lies between ***'(sr)th'*** row and ***'(mrow – 1)th'*** row and the value of ***'j'*** lies between ***'(mcol)th'*** column and ***'(sc)th'*** column, ***'b'*** is appended in the list ***'SQu'***.

The area from the ***'(mrow)th'*** row to ***'(er)th'*** row and from the ***'(sc)th'*** column to ***'(mcol – 1)th'*** column forms the ***'d'*** sub-quadrant. If the value of ***'i'*** lies between ***'(mrow)th'*** row and ***'(er)th'*** row and the value of ***'j'*** lies between ***'(sc)th'*** column and ***'(mcol – 1)th'*** column, ***'d'*** is appended in the list ***'SQu'***.

The area from the ***'(mrow)th'*** row to ***'(er)th'*** row and from the ***'(mcol)th'*** column to ***'(ec)th'*** column forms the ***'c'*** sub-quadrant. If the value of ***'i'*** lies between ***'(mrow)th'*** row and ***'(er)th'*** row and the value of ***'j'*** lies between ***'(mcol)th'*** column and ***'(ec)th'*** column, ***'c'*** is appended in the list ***'SQu'***.

Remaining_Part Module

The format of this module is Remaining_Part (quad). This module is applied to all sixteen quadrants. If Get_Pattern_Left or Get_Pattern_Right is unable to scan some continuous trace of black pixels of a particular quadrant that is stored in ***'quad',*** then to have all those missed areas that are acquired by a continuous trace of black pixels of ***'quad'***, the Remaining_Part has been used. First, this module checks if the names of all the sub-quadrants are found in the ***'LSubQ'*** list. If any of the names of the sub-quadrants is not found then that sub-quadrant is checked for the presence of a continuous trace of black pixels and if it is found then the name of the sub-quadrant is added in ***'LSubQ'***. If the names of all the sub-quadrants that have black pixels are present at least once in ***'LSubQ'*** then, Remaining_Part stops executing.

The final value returned by this module is stored in ***'LSubQ'*** of either Get_Pattern_Left or the Get_Pattern_Right module. The final value that is found in the list, ***'LsubQ'*** is the whole feature that is extracted from a particular quadrant of an alphabet.

Write_Excel Module

This module first converts the final features stored in ***'LSubQ'*** to string format by concatenating all the elements of ***'LSubQ'***. Then the final feature in string format has been pushed into the ***'(DItem)th'*** sheet, ***'(DInItem)th'*** row, and ***'(QNo)th'*** column of **'DictionaryFeatures16.xlsx'**. The path of the excel file is stored in the ***'DPath'***. The ***'SName'*** is the file name and it is pushed into the seventeenth column of the ***'(DInItem)th'*** row of the ***'(DItem)th'*** sheet of the excel file. For example, the ಅ alphabet in font sizes 18, 20, 22, 24, 26, 28, 36, 48, and 72 is pushed into the first row of the sheet with names 1, 2, 3, 4, 5, 6, 7, 8 and 9 respectively.

Find_Common_Feature Module

Now, **'DictionaryFeatures16.xlsx'** consists of the extracted features of all alphabets in all font sizes. Sheets 1, 2, 3, 4, 5, 6, 7, 8, and 9 consist of the extracted features of all alphabets in font size 18, 20, 22, 24, 26, 28, 36, 48, and 72 respectively. The idea of this module is to find a common feature of an alphabet by applying LCS (Longest Common Subsequence) on the extracted features in all font sizes repeatedly of that alphabet. The common feature which has been found by the repeated application of LCS is pushed into an excel file identified as ***'CFeature16.xlsx'*** and this excel file consists of only one sheet.
'sr16' is initialized to 1.
'sh16' is initialized to 1.
'sheet' is initialized to 9.
'row16' is the number of rows in **'DictionaryFeatures16.xlsx'**.

***Algorithm*:**

Find_Common_Feature()
1. REPEAT STEP 2 WHILE sr16 < row16
2. REPEAT STEP FROM 3 TO 21 WHILE sh16 <=sheet
3. APPEND the extracted feature in the first, second, third, fourth, fifth, sixth, seventh, eighth, ninth, tenth, eleventh, twelfth, thirteenth, fourteenth, fifteenth, and sixteenth column of the ***'sr16th'*** row of ***'sh16th'*** sheet of **'DictionaryFeatures16.xlsx'** in ***'FiQuList'***, ***'SeQuList'***, ***'ThQuList'***, ***'FoQuList'***, ***'FifQuList'***, ***'SiQuList'***, ***'SevQuList'***, ***'EiQuList'***, ***'NiQuList'***, ***'TeQuList'***, ***'EleQuList'***, ***'TweQuList'***, ***'ThiQuList'***, ***'FouQuList'***, ***'FiftQuList'*** and ***'SixQuList'*** respectively.

4. Text1 = Create_LCS(FiQuList)
5. Text2 = Create_LCS(SeQuList)
6. Text3 = Create_LCS(ThQuList)
7. Text4 = Create_LCS(FoQuList)
8. Text5 = Create_LCS(FifQuList)
9. Text6 = Create_LCS(SiQuList)
10. Text7 = Create_LCS(SevQuList)
11. Text8 = Create_LCS(EiQuList)
12. Text9 = Create_LCS(NiQuList)
13. Text10 = Create_LCS(TeQuList)
14. Text11 = Create_LCS(EleQuList)
15. Text12 = Create_LCS(TweQuList)
16. Text13 = Create_LCS(ThiQuList)
17. Text14 = Create_LCS(FouQuList)
18. Text15 = Create_LCS(FiftQuList)
19. Text16 = Create_LCS(SixQuList)
20. WRITE the contents of ***'Text1'***, ***'Text2'***, ***'Text3'***, ***'Text4'***, ***'Text5'***, ***'Text6'***, ***'Text7'***, ***'Text8'***, ***'Text9'***, ***'Text10'***, ***'Text11'***, ***'Text12'***, ***'Text13'***, ***'Text14'***, ***'Text15'*** and ***'Text16'*** in the first, second, third, fourth, fifth, sixth, seventh, eighth, ninth, tenth, eleventh, twelfth, thirteenth, fourteenth, fifteenth and sixteenth column of ***'sr^{th}'*** row of the excel file, ***'CFeature16.xlsx'*** respectively.
21. INITIALIZE ***'Text1'***, ***'Text2'***, ***'Text3'***, ***'Text4'***, ***'Text5'***, ***'Text6'***, ***'Text7'***, ***'Text8'***, ***'Text9'***, ***'Text10'***, ***'Text11'***, ***'Text12'***, ***'Text13'***, ***'Text14'***, ***'Text15'*** and ***'Text16'*** to the empty string.
22. EXIT

a) Create_LCS (li)

This procedure takes a list ***'li'*** as input. This list consists of the extracted features of a quadrant of an alphabet in all font sizes. Every iteration calls Find_LCS to find the LCS of two strings that have been passed as parameters and the result is stored in ***'SText'***. At last, the contents of ***'SText'*** are returned to Find_Common_Feature.

***Algorithm*:**

1. f=0
2. SText = li[0]
3. REPEAT STEP 4 WHILE f < LENGTH(li)
4. SText = Find_LCS(SText, li[f + 1])
5. RETURN SText
6. EXIT

b) Find_LCS (Str1, Str2)

Find_LCS calculates and returns the LCS (Longest Common Subsequence) of ***'Str1'*** and ***'Str2'***. The format of this procedure is Find_LCS(Str1, Str2).

The aim is to find the LCS of ***'Str1'*** and ***'Str2'***. If both ***'Str1'*** and ***'Str2'*** are equal to null then ***'lss'*** is assigned null and the same is returned. If ***'Str1'*** is not equal to null and ***'Str2'*** is equal to null then ***'Str1'*** is stored in ***'lss'*** and the value of ***'lss'*** is returned. If ***'Str1'*** is equal to null and ***'Str2'*** is not equal to null then ***'Str2'*** is stored in ***'lss'*** and the value of ***'lss'*** is returned. If ***'Str1'*** and ***'Str2'*** are equal and none of them are equal to null values then ***'Str1'*** is stored in ***'lss'*** and it is returned. If neither of the above conditions is true then the LCS of ***'Str1'*** and ***'Str2'*** is found and it is stored in ***'rev'*** and returned. To find the LCS, the two arrays ***'LcsF'*** and ***'bb'*** are used. The array ***'LcsF'*** stores the length of the LCS and the array ***'bb'*** stores the information about the traversing direction of the LCS. ***'sl'***, ***'up'*** and ***'lo'*** denote ***'traversing diagonally'***, ***'traversing in upward direction'*** and ***'traversing in left direction'*** respectively. After all the values of ***'LcsF'*** and ***'bb'*** are found, both arrays are traversed from the bottom-most corner and right-most side to get the value of **ii** and **jj** where ***LcsF[ii][jj] = MaxVal and bb[ii][jj] = 'sl'*** and for each ***'sl'*** in ***'bb'*** array, the common item in both the strings (Str1 and Str2) is added in ***'lss'***. The ***'MaxVal'*** is the maximum length of LCS in '***LcsF'***. At last ***'lss'*** is reversed and the resultant LCS is stored in ***'rev'***.

***Algorithm*:**

Find_LCS (Str1, Str2)

1. IF Str1 = NULL AND Str2 = NULL THEN DO FROM STEP 2 TO 3
2. lss = NULL
3. RETURN ***'lss'***
4. ELSE IF Str1 != NULL AND Str2 = NULL THEN DO FROM STEP 5 TO 6
5. lss = Str1
6. RETURN ***'lss'***
7. ELSE IF Str1 = NULL AND Str2 != NULL THEN DO FROM STEP 8 TO 9
8. lss = Str2
9. RETURN ***'lss'***
10. ELSE IF Str1 = Str2 THEN DO FROM STEP 11 TO 12
11. lss = Str1
12. RETURN ***'lss'***.
13. ELSE DO FROM STEP 14 TO 46
14. m = LENGTH(Str1)
15. n = LENGTH(Str2)
16. INITIALIZE the array ***'LcsF'*** with dimensions+ 1, n + 1) to zero.
17. INITIALIZE the array ***'bb'*** with dimensions (m + 1, n + 1) to zero.
18. ii = 0
19. jj = 0
20. REPEAT STEP 21 WHILE ii < (m + 1)
21. REPEAT FROM STEP 22 TO 30 WHILE jj <(n + 1)

22. IF Str1[ii – 1] = Str2[jj – 1] THEN DO FROM STEP 23 TO 24
23. LcsF[ii][jj] = LcsF[ii][jj] + 1
24. bb[ii][jj] = ***'sl'***
25. ELSE IF LcsF[ii – 1][jj] >= LcsF[ii][jj – 1] THEN DO STEP 26 TO 27
26. LcsF[ii][jj] = LcsF[ii – 1][jj]
27. bb[ii][jj] = ***'up'***
28. ELSE DO STEP 29 TO 30
29. LcsF[ii][jj] = LcsF[ii][jj – 1]
30. bb[ii][jj] = ***'lo'***
31. Search the maximum value in the array ***'LcsF'*** and store it in ***'MaxVal'***.
32. Traverse the array ***'LcsF'*** from the right-most side and bottom-most corner of the array and find the value of ii and jj in the array where LcsF[ii][jj] = *MaxVal* and bb[ii][jj] = ***'sl'***.
33. After values of ii and jj are found for LcsF[ii][jj] = *MaxVal* and bb[ii][jj] = ***'sl'***, DO STEP 34
34. REPEAT FROM STEP 35 TO 44 WHILE ii > 0 AND jj > 0
35. IF bb[ii][ii] = ***'sl'*** THEN DO FROM STEP 36 TO 38
36. APPEND the value in Str1[ii][jj] in the list ***'lss'***.
37. ii = ii – 1
38. jj = jj – 1
39. ELSE IF bb[ii][jj] = ***'up'*** DO FROM STEP 40 TO 41
40. ii = ii – 1
41. jj = jj
42. ELSE IF bb[ii][jj] = ***'lo'*** DO FROM STEP 43 TO 44
43. ii = ii
44. jj = jj – 1
45. REVERSE the elements of ***'lss'*** and store them in ***'rev'***.
46. RETURN ***'rev'***.
47. EXIT

Group_Vacant_Quads Module

This module deals with grouping the extracted features of Odia alphabets according to the vacant quadrants. The concept is that, when an Odia alphabet image is divided into sixteen quadrants, some quadrants do not have any black pixels. These quadrants are known as ***vacant quadrants***. These vacant quadrants produce no extracted feature. There may be alphabets that have more than one vacant quadrant. Hence, the grouping has been done based on the first vacant quadrant that is found. This grouping has been done for making the searching process to find a correct match for the input image present in the ***'alphabets'*** directory less time-consuming. The input to this module is the excel file, ***'CFeature16.xlsx'***. This excel file has only one sheet and sixteen columns. The columns of the excel file, ***'CFeature16.xlsx'*** are accessed one at a time and if in any row of that column, a null value is found then a new sheet is created in an excel file, ***'VacantQuadrants.xlsx'*** and all the sixteen values of that row are written in that sheet. For example, the ***'1st'*** column has been traversed for the total number of rows in the excel file, ***'CFeature16.xlsx'***. Suppose, in the ***'4th'*** row, the ***'1st'*** and ***'7th'*** column consists of null value (due to a vacant quadrant), the grouping is done based on the ***'1st'*** column with a null value as it appeared before the ***'7th'*** column in the iteration of columns. The features of Odia alphabets having a null value in the 1st, 2nd, 3rd, 4th, 5th, 6th, 7th, 8th, 9th, 10th, 11th, 12th, 13th, 14th, 15th, and 16th quadrant, those features are put in the sheet named as 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 and 16 of the excel file ***'VacantQuadrants.xlsx'***.

Hence, the final output of the ***'Build_Dictionary'*** is the ***'VacantQuadrants.xlsx'*** excel file which consists of the common feature for each alphabet in different sheets**.** For example, the common features of the alphabet ଅ are found in the sheet named 11 and they are:

1st – ddddddddbbbbbbbbbaabbbc
2nd – aaaaaaabbbbbbbbccccccddaaaaa
3rd – aaaaaaabccccccccaaaaaaaadccccccccc
4th – cccccbbbbb
5th – aaaaabbbbbb
6th – ddddddccccccccaabbb
7th – ddddddbbbbbbbbbcaaaa
8th – cccccbbbbbb
9th – bbbbbbbaccccdddd
10th – aaaaaaabbbb
11th – null value
12th – ccccccbbbbbbdccccccccc
13th – aaaabbbbccccccaaaaadccccccccc
14th – cccccccdd
15th – ccddddddddd
16th – ccccccccaaaaaaaaabbbbbbbbbaaaddd

Segment_Line

This is the second part of the system. This part takes an image ***'IM'*** which has been created by scanning or converting a document consisting of Odia alphabets into an image by using the software. Each document is written using the font family **'AkrutiOriAshok-99'** in a particular font size in bold style. The font sizes that have been considered are 18, 20, 22, 24, 26, 28, 36, 48, and 72.

The ***'IM'*** is given as input to the Preprocessing module. This module converts ***'IM'*** to a grayscale image and the result is stored in ***'GY'***. The ***'GY'*** image is converted to a binary image and the result is stored in ***'BIN'***. The pixels that consist of the Odia alphabets are represented as 0 and other pixels are represented as 1.

The image ***'BIN'*** is given as input to the Remove Noise module, where it goes through three phases: Phase I – Remove Boundary Spaces, Phase II-Remove Noise Row Wise, and Phase III- Remove Noise Column Wise for noise removal and provides the resultant binary image named, ***'NoFr'***.

The image ***'NoFr'*** is given as input to the Line Segmenting module. In this module, total counts of black pixels are calculated for each row and they are stored in the list ***'BCt'***. The list ***'BCt'*** is traversed from the start to end and the indices of the non-zero items which are preceded by an item equal to 0 are stored in a list ***'Tpl'***. The list ***'BCt'*** is again traversed from the end to the start and the indices of non-zero items that are succeeded by 0 are stored in a list ***'Bopl'***. Then the list ***'Bopl'*** is reversed and stored in ***'BoplRe'***. Then the lists ***'Tpl'*** and ***'BoplRe'*** are merged and stored in the list ***'Finl'***. At last, the list ***'Finl'*** is iterated to get the required line segments and the line segments are stored in the directory ***'Line Segments'***.

Character_Segment

This is the third part of the system in which individual Odia alphabets are segmented from the line segments. This part takes the directory ***'Line Segments'*** as input. The idea is to calculate the total count of black pixels in each column and store it in the list ***'BcntList'.*** By traversing the list both from start to end and from end to start, two lists are formed and they are ***'BcntRi'*** and ***'BcLiRev'***. The lists, ***'BcLiRev'*** and ***'BcntRi'*** consist of the start and end x-coordinates of an alphabet in the line segment respectively. Both lists are merged to form the final list named, ***'BcFinal'***. The list, ***'BcFinal'*** is traversed to get the segments of alphabets and they are stored in the directory ***'alphabets'***. The line segments are accessed one at a time and stored in ***'Lseg'***. The ***'Lseg'*** is converted to gray-scale image ***'GLseg'*** and the ***'GLseg'*** is converted to binary image ***'BiGLseg'***. ***'Lrow'*** is the number of rows of the binary image ***'BiGLseg'***.
'Lcol' is the number of columns of the binary image ***'BiGLseg'***.

***Algorithm*:**

1. The image ***'Lseg'*** is converted to a grayscale image and stored in ***'GLseg'***.
2. The image ***'GLseg'*** is converted to a binary image and stored in ***'BiGLseg'***.
3. The boundary spaces that encapsulate the whole text in ***'BiGLseg'*** are removed using the Remove Boundary Spaces of Remove Noise module of (Kar and Pradhan, 2021).
4. INITIALIZE ***'irr'*** and ***'icc'*** to 0.
5. REPEAT FROM STEP 6 TO 9 WHILE icc < Lcol
6. REPEAT FROM STEP 7 TO 8 WHILE irr < Lrow
7. IF BiLseg[irr][icc] = 0 THEN DO STEP 8
8. Bcnt = Bcnt + 1
9. APPEND ***'Bcnt'*** in the list ***'BcntList'***.
10. INITIALIZE ***'bii'*** to 0.
11. REPEAT FROM STEP 12 TO 13 WHILE bii < LENGTH (BcntList) - 1
12. IF BcntList[bii] != 0 AND BcntList[bii + 1] = 0 THEN DO STEP 13
13. APPEND ***(bii + 1)*** in the list ***'BcntRi'***.
14. INITIALIZE ***'bii'*** to LENGTH (BcntList).
15. REPEAT FROM STEP 16 TO 17 WHILE bii > 1
16. IF BcntList[bii] != 0 AND BcntList[bii – 1] = 0 THEN DO STEP 17
17. APPEND ***(bii - 1)*** in the list ***'BcntLi'***.
18. REVERSE the list ***'BcntLi'*** and store it in the list ***'BcLiRev'***.
19. Merge the lists, ***'BcntRi'*** and ***'BcLiRev'*** and store them in the list ***'BcFinal'.***
20. Iterate through the list ***'BcFinal'*** to get the segment of the individual alphabet and store it in the directory ***'alphabets'***.
21. EXIT

Recognise_Character

This is the fourth part of the system in which the images of the Odia alphabets present in the directory ***'alphabets'*** are accessed for feature extraction and then the features are searched in the excel file, '***VacantQuadrants.xlsx***' to get a correct match. Figure 3 shows the flow of control when the image ***'IM'*** is given as input to the system. The ***'IM'*** is an image of a document that has been written in the font family **'AkrutiOriAshok-99'** in a particular font size. The ***'IM'*** is given as input to the Segment_Line part where it undergoes certain steps to form line segments and these line segments are stored in the ***'Line Segments'*** directory. Then each line segment ***'LSeg'*** from ***'Line Segments'*** is accessed and given as input to the Character_Segment part for the segmentation of the Odia alphabet from the line segment. All segmented Odia alphabets are stored in the directory, ***'alphabets',*** and each image of an Odia alphabet, ***'Inp'*** undergoes the modules like PreProcessing, Find_Direction, Get_Pattern_Left, Get_Pattern_Right, Sub_Quad_Pattern, Remaining_Part, and Write_Excel for extraction of features, and the extracted features are written in an excel file, ***'InputFile16.xlsx'***.

Pre-Processing Module

Each segmented Odia alphabet present in the directory ***'alphabets'*** is given as input to this module and is stored in ***'Inp'***.

Algorithm:

1. The image '***Inp'*** is converted to a gray-scale image and stored in '***BInp'***.
2. The white spaces that enclose the text in the image '***BInp'*** are removed using RemoveBoundarySpaces

of the RemoveNoise module of (Kar and Pradhan, 2021) and the result is stored in '***CInp'***.

3. The image '***CInp'*** is transformed into a size of 64 x 64 by using inter-cubic interpolation and the result is stored in '***DInp'***.
4. The image '***DInp'*** is transformed into a binary image and stored in '***EInp'*.**
5. The ***'EInp'*** is partitioned into four equal slices, both in horizontal and vertical directions. In this way, this image ***'EInp'*** is partitioned into sixteen equal quadrants. The dimension of this image ***'EInp'*** is rowIn16 x colIn16 (rowIn16 = 64 and colIn16 = 64), where ***'rowIn16'*** is the number of rows and ***'colIn16'*** is the number of columns.
6. The values for dividing the image ***'EInp'*** horizontally are calculated in the following ways:

$$rowIn1 = \left\lfloor \frac{rowIn16}{4} \right\rfloor$$

$$rowIn2 = 2 * rowIn1$$

$$rowIn3 = rowIn2 + rowIn1$$

7. The values for dividing the image ***'EInp'*** vertically are calculated in the following ways:

$$colIn1 = \left\lfloor \frac{colIn16}{4} \right\rfloor$$

$$colIn2 = 2 * colIn1$$

$$colIn2 = 2 * colIn1$$

8. The sixteen quadrants are found from the ***'EInp'*** by using the values calculated in steps **6** and **7**. The sixteen quadrants are named ***'FirstIn'***, ***'SecondIn'***, ***'ThirdIn'***, ***'FourthIn'***, ***'FifthIn'***, ***'SixthIn'***, ***'SeventhIn'***, ***'EighthIn'***, ***'NinthIn'***, ***'TenthIn'***, ***'EleventhIn'***, ***'TwelvthIn'***, ***'ThirteenthIn'***, ***'FourteenthIn'***, ***'FifteenthIn'*** and ***'SixteenthIn'***.
FirstIn = EInp[0:rowIn1, 0:colIn1]
SecondIn = EInp[0:rowIn1, (colIn1 + 1):colIn2]
ThirdIn = EInp[0:rowIn1, (colIn2 + 1):colIn3]
FourthIn = EInp[0:rowIn1, (colIn3 +1):colIn16]
FifthIn = EInp[(rowIn1 + 1):rowIn2, 0:colIn1]
SixthIn = EInp[(rowIn1 + 1):rowIn2, (colIn1 + 1):colIn2]
SeventhIn = EInp[(rowIn1 + 1):rowIn2, (colIn2 + 1):colIn3]
EighthIn = EInp[(rowIn1 + 1):rowIn2, (colIn3 + 1):colIn16]
NinthIn = EInp[(rowIn2 + 1):rowIn3, 0:colIn1]
TenthIn = EInp[(rowIn2 + 1):rowIn3, (colIn1 + 1):colIn2]
EleventhIn = EInp[(rowIn2 + 1):rowIn3, (colIn2 + 1):colIn3]
TwelfthIn = EInp[(rowIn2 + 1):rowIn3, (colIn3 +1):colIn16]
ThirteenthIn = EInp[(rowIn3 + 1):rowIn16, 0:colIn1]
FourteenthIn = EInp[(rowIn3 + 1):rowIn16, (colIn1 + 1):colIn2]
FifteenthIn = EInp[(rowIn3 + 1):rowIn16, (colIn2 + 1):colIn3]
SixteenthIn = EInp[(rowIn3 + 1):rowIn16, (colIn3 + 1):colIn16]
9. Call ***Find_Direction(QNo, quad, DItem, DInItem, DPath, SName)*** for the quadrants ***FirstIn***, ***SecondIn***, ***ThirdIn***, ***FourthIn***, ***FifthIn***, ***SixthIn***, ***SeventhIn***, ***EighthIn***, ***NinthIn***, ***TenthIn***, ***EleventhIn***, ***TwelvthIn***, ***ThirteenthIn***, ***FourteenthIn***, ***FifteenthIn***, and ***SixteenthIn*** where ***QNo*** is the number of quadrants among the sixteen quadrants. Here, QNo = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 ***quad*** can be ***FirstIn***, ***SecondIn***, ***ThirdIn***, ***FourthIn***, ***FifthIn***, ***SixthIn***, ***SeventhIn***, ***EighthIn***, ***NinthIn***, ***TenthIn***, ***EleventhIn***, ***TwelvthIn***, ***ThirteenthIn***, ***FourteenthIn***, ***FifteenthIn***, and ***SixteenthIn.*** ***'DItem'*** and ***'DInItem'*** are constants and are set to 1 as at a particular time only one Odia alphabet is accessed.
SName is the name of the image file present in ***'Inp'.***

Find_Direction Module

The procedure followed in this section is the same as described in the Find_Direction module of ***'Build_Dictionary'*** except that the procedure of the Find_Direction is performed for the quadrants ***'FirstIn'***, ***'SecondIn'***, ***'ThirdIn'***, ***'FourthIn'***, ***'FifthIn'***, ***'SixthIn'***, ***'SeventhIn'***, ***'EighthIn'***, ***'NinthIn'***, ***'TenthIn'***, ***'EleventhIn'***, ***'TwelvthIn'***, ***'ThirteenthIn'***, ***'FourteenthIn'***, ***'FifteenthIn'*** and ***'SixteenthIn'***. Here ***'DItem'*** and ***'DInItem'*** are constants and are set to 1 as at a particular time only one Odia alphabet is accessed. The ***'DPath'*** parameter holds the absolute path of the excel file named ***'InputFile16.xlsx'*** and in this file, features of all the sixteen quadrants of the input image are written. The features of ***'FirstIn'***, ***'SecondIn'***, ***'ThirdIn'***, ***'FourthIn'***, ***'FifthIn'***, ***'SixthIn'***, ***'SeventhIn'***, ***'EighthIn'***, ***'NinthIn'***, ***'TenthIn'***, ***'EleventhIn'***, ***'TwelvthIn'***, ***'ThirteenthIn'***, ***'FourteenthIn'***, ***'FifteenthIn'*** and ***'SixteenthIn'*** are pushed into the first, second, third, fourth, fifth, sixth, seventh, eighth, ninth, tenth, eleventh, twelfth, thirteenth, fourteenth, fifteenth and sixteenth column of ***'InputFile16.xlsx'*** respectively.

Get_Pattern_Left Module

The procedure followed in this section is the same as described in the Get_Pattern_Left module of ***'Build_Dictionary'*** except that the procedure here is applied to the quadrants ***'FourthIn'***, ***'EighthIn'***, ***'NinthIn'***, ***'EleventhIn'***, ***'TwelvthIn'***, ***'FourteenthIn'***, ***'FifteenthIn'*** and ***'SixteenthIn'***.

Get_Pattern_Right Module

The procedure followed in this section is the same as described in the Get_Pattern_Right module of ***'Build_Dictionary'*** except that the procedure here is applied to the quadrants ***'FirstIn'***, ***'SecondIn'***, ***'ThirdIn'***, ***'FifthIn'***, ***'SixthIn'***, ***'SeventhIn'***, ***'TenthIn'*** and ***'ThirteenthIn'***.

Sub_Quad_Pattern Module

The procedure followed in this section is the same as described in the Sub_Quad_Pattern module of ***'Build_Dictionary'*** except that the procedure is applied to the quadrants '***FirstIn'***, '***SecondIn'***, ***'ThirdIn'***, ***'FourthIn'***, ***'FifthIn'***, ***'SixthIn'***, ***'SeventhIn'***, ***'EighthIn'***, ***'NinthIn'***, ***'TenthIn'***, ***'EleventhIn'***, ***'TwelvthIn'***, ***'ThirteenthIn'***, ***'FourteenthIn'***, ***'FifteenthIn'*** and ***'SixteenthIn'***.

Remaining_Part Module

The procedure followed in this section is the same as described in the Remaining_Part module of ***'Build_Dictionary'*** except that the procedure is applied to the quadrants '***FirstIn'***, '***SecondIn'***, ***'ThirdIn'***, ***'FourthIn'***, ***'FifthIn'***, ***'SixthIn'***, ***'SeventhIn'***, ***'EighthIn'***, ***'NinthIn'***, ***'TenthIn'***, ***'EleventhIn'***, ***'TwelvthIn'***, ***'ThirteenthIn'***, ***'FourteenthIn'***, ***'FifteenthIn'*** and ***'SixteenthIn'***.

Write_Excel Module

The procedure followed in this section is the same as described in the Write_Excel module of ***'Build_Dictionary'*** except that the procedure here is applied for the features extraction from the quadrant '***FirstIn'***, '***SecondIn'***, ***'ThirdIn'***, ***'FourthIn'***, ***'FifthIn'***, ***'SixthIn'***, ***'SeventhIn'***, ***'EighthIn'***, ***'NinthIn'***, ***'TenthIn'***, ***'EleventhIn'***, ***'TwelvthIn'***, ***'ThirteenthIn'***, ***'FourteenthIn'***, ***'FifteenthIn'*** and ***'SixteenthIn'*** are written in an excel file named as ***'InputFile16.xlsx'***. The absolute path of ***'InputFile16.xlsx'*** is stored in the ***'DPath'*** parameter and the file name of the input image is stored in ***'SName'***. The value in the ***'SName'*** parameter is written in the seventeenth column of ***'InputFile16.xlsx'.***

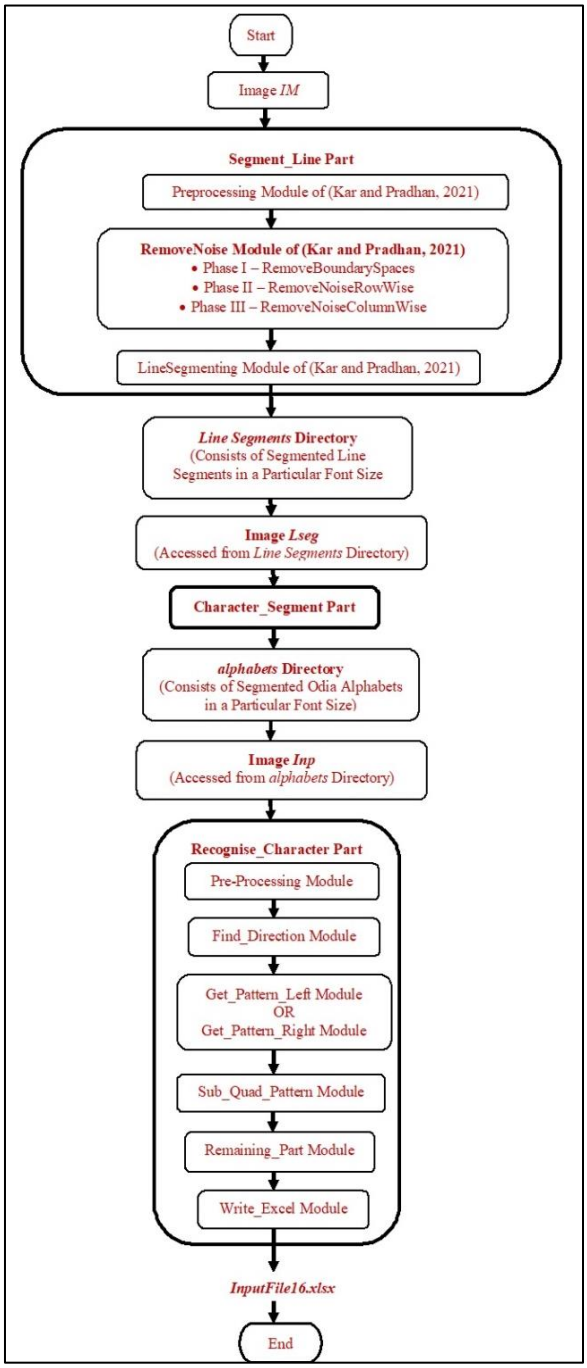

Fig. 3: Extraction of feature from input document image **IM**

Hence, the feature extracted from any of the above sixteen quadrants and the value in the ***'SName'*** parameter is written in the first, second, third, fourth, fifth, sixth, seventh, eighth, ninth, tenth, eleventh, twelfth, thirteenth, fourteenth, fifteenth, sixteenth and seventeenth column of the first row of the excel file, '***InputFile16.xlsx'*** respectively and there is only one sheet present in the excel file. For example, the final features for the input image ***'Inp'*** are:

1st – ddddddddbbbbbbbbbaaabbbbbbbbccddddddda
2nd – aaaaaaabbbbbbbbbccccccdddaaaaa
3rd – aaaaaaabbbbbbccccccccaaaaaaaadccccccccc
4th – ccccccccbbbbbbbb
5th – aaaaaabbbbbbccc
6th – ddddddddccccccccbbaaaabbbbb
7th – dddddaabbbbbbbbbbcdaaaaaaa
8th – ccccccccbbbbbbbb
9th – bbbbbbbbaaccccddddd
10th – aaaaaaabbbb
11th – null value
12th – ccccccccbbbbbbbbddddccccccccc
13th – aaaabbbbbccccccccbbaaaaadccccccccc
14th – ccccccccdddaaaaaa
15th – bbbbbccddddddddddd
16th–ccccccccaaaaaaaaaabbbbbbbbbaaaaaaaaddddddddaccccc

Search_For_Match

This module consists of two sub-modules, ***'Check_Common'*** and ***'Match_Common'***. The submodule, ***'Check_Common'*** takes the features extracted from the input image, '***Inp'*** which are stored in the excel file, ***'InputFile16.xlsx'*** as input. Figure 4 shows the overall process of finding a correct match. In the left part of Fig. 4, the extracted features in ***'InputFile16.xlsx'*** are accessed and searched in the sheets of the excel file, ***'VacantQuadrants.xlsx'***. If any of the sixteen columns in ***'InputFile16.xlsx'*** has a null value, then the sheet named with the column number having null is accessed. For example, if the 6th and 10th columns consist of null values, the sheet with the name ***'6'*** in ***'VacantQuadrants.xlsx'*** is searched for finding a correct match using the Check_Common sub-module. If multiple matches are found in the 6th sheet then, the Match_Common sub-module is applied. After this, if a correct match is not found then a sheet named ***'10'*** is searched using the Check_Common sub-module. Again, if multiple matches are found then the Match_Common sub-module is applied to get a correct match.

The right part of Fig. 4 shows the process if none of the columns of ***'InputFile16.xlsx'*** have null values. If none of the columns of ***'InputFile16.xlsx'*** have null values then the extracted features in ***'InputFile16.xlsx'*** are searched in ***'CFeature16.xlsx'*** using the Check_Common sub-module**.** As explained earlier in the Find_Common_Feature module of ***'Build_Dictionary'***, the ***'CFeature16.xlsx'*** is a repository of the common features of all Odia alphabets. After the application of the Check_Common sub-module, if multiple matches are found then the Match_Common sub-module is applied to get a correct match.

Check_Common

The common features present in all the columns in each row of ***'CFeature16.xlsx'*** are traversed to find whether the common features are found in the corresponding columns of ***'InputFile16.xlsx'***. For this, Check_Common has been used. The format of Check_Common is Check_Common (String1, String2). If the features in all the columns of ***'InputFile16.xlsx'*** are present in the respective columns of ***'CFeature16.xlsx'*** in a particular row, then the parameter ***'P1'*** is made equal to 1 otherwise it is made equal to 0. If P1 = 1 then the parameter ***'P3'*** is incremented by 1. For example, if the value of P3 is 16 after all the columns of ***'InputFile16.xlsx'*** have been checked with the corresponding columns for a row ***'r'*** in ***'CFeature16.xlsx'***, then the file name is retrieved from the seventeenth column of the row ***'r'*** in ***'CFeature16.xlsx'.*** The file name from the seventeenth column of row ***'r'*** is added to the list ***'MFirst'*** and the row number of row ***'r'*** is added to the list ***'MRow'***. In some scenarios, the list ***'MFirst'*** has more than one correct match and in these scenarios ***'Match_Common'*** is called.

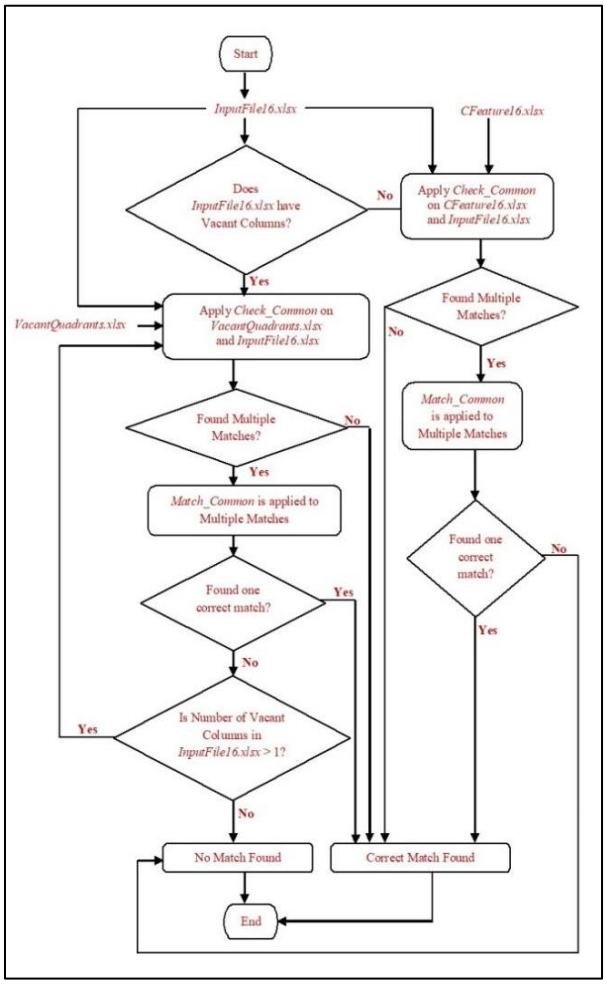

Fig. 4: Process of finding the correct match

Match_Common

If there is more than one value in the list ***'MFirst'*** then, this module is called. In this module, for each row number in ***'MRow',*** the common features of all the columns are retrieved from ***'CFeature16.xlsx'*** and appended in the list ***'QList'***. The features present in all the sixteen columns of ***'InputFile16.xlsx'*** are retrieved and appended in the list '***InpPattern'.*** Then the LCS of the two strings, ***'Str1'*** and ***'Str2'*** is found where Str1 = QList[ee] and Str2 = InpPattern[ee] where, ee = 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16 and the resultant LCS is matched with ***'Str2'***. If there is a match with ***'Str2'*** then the parameter ***'P2'*** is set to 1, otherwise 0. If P2 = 1 then, the parameter ***'P4'*** is incremented by 1. These steps are carried out for each item present in the list ***'MRow'***. If P4 = 16 for the row number in ***'MRow',*** then the index of that row number is retrieved and stored in ***'mrw'***. The file name at the index ***'mrw'*** in the list ***'MFirst'*** is copied to the list ***'MSecond'***. The list ***'MSecond'*** consists of the correct match for the input image.

Results and Discussion

Though the proposed system in this study follows some parts of the system described in (Kar and Pradhan, 2022), some other parts are newly added to the system. The system described in this study differs from the system described by (Kar and Pradhan, 2022) in the number of parts, division of the image of the Odia alphabet, direction which is followed to get a continuous trace of the black pixel, finding features of the images in ***'Dictionary'***, input to the system and the correctness accuracy in the recognition of Odia alphabets; and the all the differences of the proposed system and the system described in the system (Kar and Pradhan, 2022) are tabulated in Table 1.

The proposed system has been implemented using Lenovo Ideapad 310 Laptop, 4GB RAM, Intel(R) Core (TM) i5-7200U CPU @ 2.50GHz 2.70 GHz, and the JetBrains PyCharm Community Edition 2019.1.

Correctness Accuracy

For testing, 100 documents consisting of Odia alphabets written in the font family, **'AkrutiOriAshok-99'** in a particular font size in bold style are taken. The font sizes that are considered are 18, 20, 22, 24, 26, 28, 36, 48, and 72. For each font size, 100 scanned documents or documents converted to images by using the software are created. In this way, 900 images of documents are tested. The groups are created for each set of 100 documents in a particular font size. The groups are named Doc18, Doc20, Doc22, Doc24, Doc26, Doc28, Doc32, Doc48, and Doc72 for the font size 18, 20, 22, 24, 26, 28, 36, 48, and 72 respectively. Hence, the proposed system has been tested on 900 document images. An image from group Doc36 as an example for testing has been shown in Fig. 5.

Table 1: Difference between the system described in this study and the system described in (Kar and Pradhan, 2022)

	Features of the system described in this study	Features the system described in (Kar and Pradhan, 2022)
1.	Both the dictionary images and input images are divided into 16 quadrants	Both dictionary images and input images were divided into 4 quadrants
2.	In the Pre-Processing module of the Build_Dictionary part, both the dictionary images and the input image are divided into 4 parts both vertically and horizontally	In the Preprocessing module of Dictionary Building, both the dictionary images and the input image was divided into 2 parts both vertically and horizontally
3.	This system consists of four parts, Build_Dictionary, Segment_ Line, Character _ Segment, and Recognise_Character	This system consists of two parts, Dictionary Building Finding Match
4.	Out of sixteen quadrants, 8 quadrants are scanned using Get_Pattern_Left and 8 quadrants are scanned using Get_Pattern_ Right in the Find_Direction module	Out of four quadrants, 1 quadrant was scanned using Getting Features Left and 3 quadrants were scanned using Getting Features Right in the Find Path module
5.	When an image of an Odia alphabet is divided into sixteen quadrants, some quadrants do not have any black pixels. These quadrants are known as Vacant Quadrants	None of the 4 quadrants were vacant
6.	Based on vacant quadrants, features of images of Odia alphabets are grouped and stored in an excel file, *VacantQuadrants.xlsx'.* This is done by the Group_Vacant_Quads module	As there were no vacant quadrants hence there was no such module
7.	For finding a correct match, the extracted feature of the input image is searched in the excel file, *'VacantQuadrants.xlsx'* and if a correct match is not found then the *'CFeature16.xlsx'* excel file is searched. There are no steps for tracing the input alphabet in another direction as by searching the above two excel files a correct match can be found	For finding a correct match, the extracted feature of the input image was searched in the excel file, *'CommonFeature.xlsx'* and if a correct match was not found then the input alphabet was traced in another direction using the module, Trace Another Direction
8.	For testing, images of documents are given as input and a correctness accuracy of 99.8% has been achieved	For testing, images of Odia alphabets were given as input and a correctness accuracy of 98.1% had been achieved

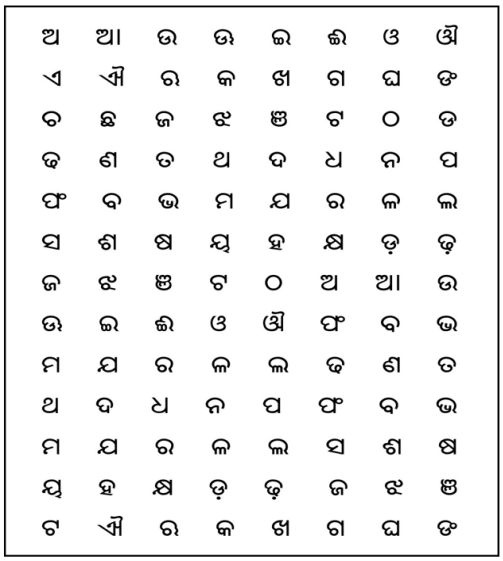

Fig. 5: Image of a Document Given for Testing

The image shown in Fig. 5 is stored in ***'IM'*** and goes through the steps that are shown in Fig. 3 for feature extraction and follows the steps shown in Fig. 4 for finding a correct match.

Each segmented Odia alphabet finds a match in the excel file, ***'VacantQuadrants.xlsx'*** using the Search_For_Match module. In the row of the excel file, ***'VacantQuadrants.xlsx'*** where a segmented Odia alphabet finds a match, a variable ***'count'*** is incremented by 1. In the same way, for each different Odia alphabet, a different variable is created and incremented by 1. In this way, there are a total of 48 variables, each variable dedicated to a particular Odia alphabet. Initially, the number of times a particular alphabet appears in the input image ***'IM'*** has been counted manually. The value of the variable for a particular alphabet is compared with the count that has been manually found for each alphabet.

This process is repeated for all 100 documents in a particular font size to find the correctness and accuracy of the recognition of the Odia alphabets in a particular font size. The correctness accuracy of recognition of Segmented Odia alphabets is calculated as:

$$Correctness\ Accuracy = \frac{Num\ Of\ Cor\ Mat * 100}{To\ Num\ Of\ Al}$$

where:

Num Of Cor Mat = Number of Correct Matches in all the input documents in a particular font size

To Num Of Al = Total Number of Alphabets in all the input documents in a particular font size

The correctness accuracy of segmentation of individual Odia alphabets is calculated as:

$$Correctness\ Accuracy = \frac{Num\ Of\ Char * 100}{To\ Num\ Of\ Char}$$

where:

Num Of Char = *Number of Correctly Segmented Alphabets in all the input documents in a particular font size*

To Num Of Char = *Total Number of Alphabets in all the input documents in a particular font size*

According to the testing results, the system has achieved an overall accuracy of 99.97 and 99.8% in the segmentation of individual Odia alphabets from line segments and recognition of segmented Odia alphabets respectively. The results related to the correctness accuracy for the recognition of segmented Odia alphabets have been shown in Fig. 6.

Comparison with other Approaches

In this section, the proposed system in this study is compared with other approaches. The comparison has been shown in Table 2.

Table 2 shows the comparison of the correctness accuracy of the recognition of alphabets in different languages with the correctness accuracy of the proposed system.

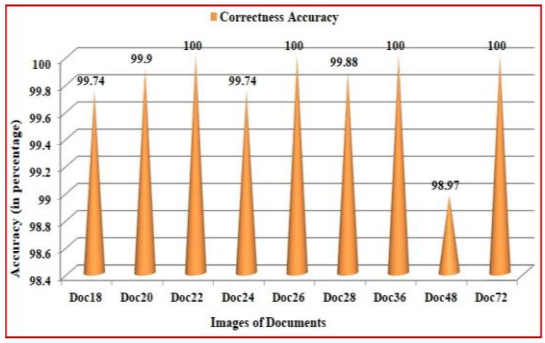

Fig. 6: Correctness Accuracy of the Recognition of Segmented Odia Alphabets

Table 2: Comparison of accuracy

Approaches	Language	Accuracy %	Accuracy achieved by the approach in this study
Accuracy achieved by other approaches			
Kar and Pradhan (2022)			
Pal *et al*. (2003)	Printed odia	98.10	
Das *et al*. (2017)	Handwritten English Numerals	94.80	99.8%
Padhi and Senapati (2013)	Printed odia	92.00	
Goswami and Mitra (2018)	Printed odia	91.30	
	Printed gujarati	96.87	

Conclusion

The system explained in this study has four parts, ***'Build_Dictionary'***,***'Segment_Line'***,***'Character_Segment'***, and ***'Recognise_Character'***. In this system, features are extracted from the Odia alphabet images present in the directory ***'Dictionary'*** and stored in the excel file, **'DictionaryFeatures16.xlsx'**. From this excel file, common features are extracted and stored in ***'CFeature16.xlsx'*** and the data in ***'CFeature16.xlsx'*** are grouped based on vacant quadrants. These groups are stored in separate sheets of an excel file, ***'VacantQuadrants.xlsx'***. The final output of the ***'Build_Dictionary'*** is the excel file, ***'VacantQuadrants.xlsx'.*** On the other side, a document image consisting of Odia alphabets is given as input to the ***'Segment_Line'*** part which helps in creating images of line segments from the input document and stores them in the ***'Line Segments'*** directory. The ***'Character_Segment'*** part creates images of individual Odia alphabets from the line segments and stores them in the directory ***'alphabets'***. One alphabet at a time is given as input to the ***'Recognise_Character'*** part where features are extracted and these features are searched in ***'VacantQuadrants.xlsx'*** to get a correct match. If a correct match is not found, then features extracted from the input image of the Odia alphabet are searched in ***'CFeature16.xlsx'*** to get a correct match. The system has achieved an overall accuracy of 99.8%.

In the future, research can be done to reduce the number of modules thereby increasing the accuracy. This system can be tested on other font families of the Odia language. The system can be tested on the distorted images of Odia alphabets to observe the amount of accuracy.

Acknowledgment

The authors acknowledge the support received from the Department of Computer Science and Applications, Utkal University, Bhubaneswar, Odisha, India.

Funding Information

There was no outside funding for the research presented in the manuscript.

Author's Contributions

Aradhana Kar: System design and acquisition of data, drafted of the article, final approval of the version of the article.

Sateesh Kumar Pradhan: System designed, analysis of data and interpretation, reviewing of the article, final approval of the version of the article.

Ethics

The article is original in its approach and all the authors have read and approved the manuscript and no ethical issues are involved.

References

Agarwal, U. (2008). Algorithms Design and Analysis. Dhanpat Rai & Co.

Beazley, D., & Jones, B. K. (2011). Python Cookbook: Recipes for Mastering Python 3. O'Reilly Media, Inc.

Bigun, J. (2006). Vision with Direction: A Systematic Introduction to Image Processing and Computer Vision. Springer. https://link.springer.com/book/10.1007/b138918

Chityala, R., & Pudipeddi, S. (2020). Image processing and acquisition using Python. Chapman and Hall/CRC.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms. MIT Press.

Das, D., Dash, R., & Majhi, B. (2017). Odia Compound Character Recognition Using Stroke Analysis. In Computational Intelligence in Data Mining. *Advances in Intelligent Systems and Computing*, Vol 556, (pp. 325-332). Springer Singapore. https://doi.org/10.1007/978-981-10-3874-7_30

Downey, A. B. (2019). Think Python. Green Peas Press. https://www.greenteapress.com/thinkpython/thinkpython.pdf

Gonzalez, R. C., & Woods, R. E. (2017). Digital Image Processing. Pearson Education India.

Gazoni, E., & Clark, C. (2022). openpyxl-A Python library to read/write Excel 2010 xlsx/xlsm files. https://openpyxl.readthedocs.io/en/stable/

Goswami, M.M., & Mitra, S.K. (2018). Printed Gujarati Character Classification Using High-Level Strokes. Proceedings of 2nd International Conference on Computer Vision & Image Processing vol 2. *Advances in Intelligent Systems and Computing*, vol 704. Springer, Singapore. https://doi.org/10.1007/978-981-10-7898-9_16

Goswami, M. M., & Mitra, S. K. (2016). Classification of printed Gujarati characters using low-level stroke features. ACM Transactions on Asian and Low-Resource Language Information Processing (TALLIP), 15(4), (pp. 1-26). https://doi.org/10.1145/2856105

Howse, J., Joshi, P. & Beyeler, M. (2016). OpenCv: Computer Vision Projects with Python. *Packt Publishing*.

Idris, I. (2011). Numpy 1.5 Beginner's Guide. Packt Publishing.

Idris, I. (2012). Numpy Cookbook. Packt Publishing.

Johansson, R. (2019). Numerical Python: Scientific Computing and Data Science Applications with Numpy, Scipy and Matplotlib. *Apress*.

Kar, A., & Pradhan, S. K. (2022) Recognition of Odia Character in an Image by Dividing the Image into Four Quadrants. In International Journal of Advanced Computer Science and Applications (IJACSA), 13(8), (pp. 116 – 129).

Kar, A., & Pradhan, S. K. (2021). A Three-Phase Noise Removal Approach to Achieve Accuracy in Line Segmentation of Odia Text. In 2021 19th OITS International Conference on Information Technology (OCIT) (pp. 54-59). IEEE. https://doi.org/10.1109/OCIT53463.2021.00022

Narasimha Karumanchi, N. K. (2017). Data Structures and Algorithms Made Easy. https://eprints.triatmamulya.ac.id/1694/

Lee, K. D. (2014). Python Programming Fundamentals. SpringerLink.

Nixon, M., & Aguado, A. (2019). Feature Extraction and Image Processing for Computer Vision. *Academic Press.*

Otsu, N. (1979). A Threshold Selection Method from Gray-Level Histograms. IEEE Transactions on Systems, Man and Cybernetics, 9(1), (pp. 62-66).

Padhi, D., & Senapati, D. (2013). Zone Centroid Distance and Standard Deviation Based Feature Matrix for Odia Handwritten Character Recognition. In Proceedings of the International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA) (pp. 649 – 658). *Springer-Verlag Berlin Heidelberg*. https://doi.org/10.1007/978-3-642-35314-7_73

Pal, U., Belaıd, A., & Choisy, C. (2003). Touching Numeral Segmentation using Water Reservoir Concept. Pattern Recognition Letters, 24(1-3), (pp. 261-272). https://doi.org/10.1016/S0167-8655(02)00240-4

Mohapatra, P. N., & Das, S. (1952). Sarbasara Byakarana. ISBN: 8186085009, Published by New Students' Store. https://drive.google.com/file/d/11RNY6zXN-NmE_GeEVDYtsKxrNghHtsF3/view

Poladi, S. R. (2018). Matplotlib 3.0 Cookbook. *Packt Publishing*.

Rao, M. (1901). Chabila Madhu Barnabodha, Odisha Book Emporium. https://drive.google.com/file/d/1djPXDN4nmoVhgAdWzNkG8Yn14nhsOb8m/view

Shaw, Z. A. (2013). Learn Python 3 the Hard Way: A Very Simple Introduction to the Terrifyingly Beautiful World of Computers and Code. Addison-Wesley.

Skiena, S. S. (2013). The Algorithm Design Manual. *Springer*.

Solem, J. E. (2012). Programming Computer Vision with Python: Tools and Algorithms for Analyzing Images. *O'Reilly Media, Inc.*

Sridhar, S. (2013). Digital Image Processing. *OXFORD University Press.*

Summerfield, M. (2009). Programming in Python 3: A Complete Introduction to the Python Language. *Addison-Wesley*.

Tripathy, N., & Pal, U. (2006). Handwriting Segmentation of Unconstrained Oriya Text. *In Sadhana*, Vol. 31, (pp. 755-769). https://doi.org/10.1007/BF02716894

Vermani, L. R., & Vermani, S. (2019). An Elementary Approach to Design and Analysis of Algorithms. *World Scientific Europe Ltd*.