# Recognition of Odia Character in an Image by Dividing the Image into Four Quadrants

Aradhana Kar, Sateesh Kumar Pradhan
Department of Computer Science & Applications
Utkal University
Bhubaneswar, Odisha, India

***Abstract*—This paper deals with optical character recognition of Odia characters written in a particular font family 'AkrutiOriAshok-99' with different font sizes 18, 20, 22, 24, 26, 28, 36, 48 and 72 in Bold style. The font 'AkrutiOriAshok-99' is a font from the typing software 'Akruti'. The basic idea behind the approach followed in this paper is the character decomposition into four quadrants and then extracting features from each quadrant. The image processing techniques like converting the image to gray, resizing of image and converting gray image to binary are used in this approach. The system explained in this paper has two major parts: DictionaryBuilding and FindingMatch. For DictionaryBuilding, dictionary of images which are created either by scanning a document or a document converted to image, both written in same font family in different sizes. The features are extracted from each image in any font size in the 'Dictionary' using Preprocessing, FindPath, GettingFeaturesLeft or GettingFeaturesRight, VisitSubQuad, RemainingSubQuad, WriteToExcel and CommonFeature modules. The part FindingMatch is responsible for finding a correct match in the dictionary for the input image. For this, FeatureExtraction and Recognition modules have been used. Longest Common Subsequence (LCS) has been used for finding the common feature in DictionaryBuilding as well as finding the correct match. A total of 1800 characters, 200 characters of each font size have been tested and 98.1% of correctness has been achieved.**

***Keywords*—Odia characters; image processing; character decomposition; machine learning; optical character recognition**

## I. Introduction

In present days, the textual data are either scanned or converted to image by using software to store the data in the form of image. It is required to recognise the characters present in the scanned document or document converted to image by using some algorithm. For recognition of characters in an image, efficiency in the segmentation of lines, words and characters should be achieved.

In Odia language, the alphabets are grouped into three categories: Swara Barna, Byanjana Barna and Atirikta Barna [1] (Fig.1). Only Chandra Bindu ( ଁ), Anusara ( ଂ) and Bisarga ( ଃ ), which are part of Byanjana Barna can be used with all the alphabets of Swara Barna, other alphabets of Byanjana Barna and all alphabets of Atirikta Barna to form words. When a Swara Barna is used with the alphabets of Byanjana Barna and Atirikta Barna, the former is used as a symbol with latter to form words. These symbols are known as Matras [2]. When a Byanjana Barna alphabet is used as a symbol with the other alphabets of Byanjana Barna, these are called Juktakhyara [2].

There are different types of software available for typing Odia language in a computer. Akruti and Microsoft Indic Language Input tool for Odia are some popularly used typing software.

This paper has concentrated on the recognition of Swara Barna, Byanjana Barna and Atirikta Barna. The alphabet is written in a document in a particular font family ***'AkrutiOriAshok-99'*** in a particular font size in bold style. The font sizes that are considered in this paper are 18, 20, 22, 24, 26, 28, 36, 48 and 72. This document is either scanned or converted to image by software. The approach described in this paper first creates a dictionary of images written in ***'AkrutiOriAshok-99'*** font family, with font sizes 18, 20, 22, 24, 26, 28, 36, 48 and 72 and in bold style. The features of these images are extracted using Preprocessing, FindPath, GettingFeaturesLeft or GettingFeaturesRight, VisitSubQuad, RemainingSubQuad modules and the extracted features are written to the excel file using WriteToExcel module. A common feature is extracted from the extracted features from the images in dictionary using CommonFeature module. For finding a correct match for the input image in the dictionary of features, FindingMatch has been used. For finding a correct match, CheckCommonFeature module of Recognition has been used. If a correct match has not been found by the CheckCommonFeature module, then MatchCommonFeature module has been used to find a correct match. If in some cases, these two modules of Recognition are unable to find a correct match, the TraceAnotherDirection module of Recognition has been used.

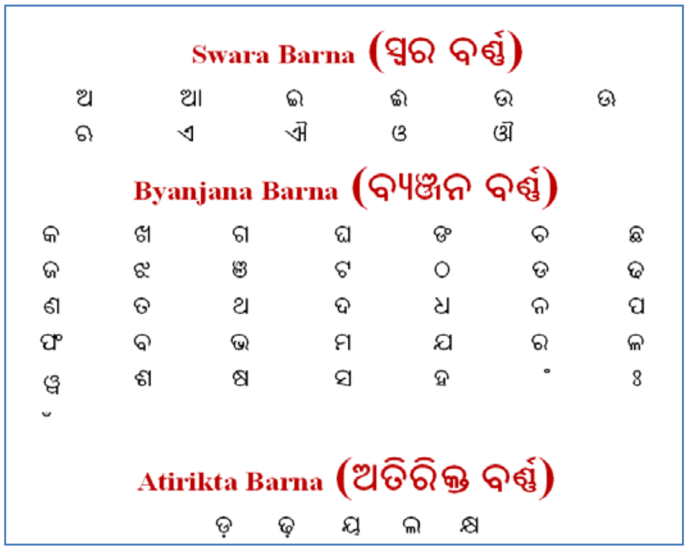

Fig. 1. Odia Alphabets.

The Preprocessing module in ***'DictionaryBuilding'*** and ***'FindingMatch'*** converts the image into gray image and then the white spaces surrounding the Odia alphabet in the gray image are removed using the Phase – 1 of RemoveNoise module of [3] (RemoveBoundarySpaces). For converting image to gray, OpenCv package of python has been used. After the elimination of white spaces from gray image, it is resized into 64 x 64 and the resultant resized image is converted to binary by using OSTU's method of thresholding [4, 5, 6]. Gray image is a type of image where intensity is stored as an 8-bit integer, hence each pixel can have intensity value ranging from 0 – 255 [7]. Binary Image is a type of image where image data is represented in terms of 0 and 1[7, 8, 9]. The basic idea for extracting features followed in this paper for DictionaryBuilding and FindingMatch is dividing the image into four quadrants and then tracing continuous path of black pixel in a particular direction in each quadrant. Experimentally, a specific direction of tracing has been agreed upon for each quadrant. The inputs to DictionaryBuilding and FindingMatch are a directory named as ***'Dictionary'*** (consists of all alphabets of Swara Barna, Byanjana Barna and Atirikta Barna of Odia language) and a directory named as ***'Input'*** (consisting of an image of Odia alphabet) respectively. The files present in ***'Dictionary'*** are accessed using os package of python [10]. The extracted features for DictionaryBuilding and FindingMatch are written in excel files, ***DictionaryFeatures.xlsx'*** and ***'InputFile.xlsx'*** respectively by using openpyxl package of python [11]. The common feature is extracted from the extracted features present in ***DictionaryFeatures.xlsx'*** by using ***Longest Common Subsequence (LCS)*** [12, 13, 14, 15, 16] and the common feature is written to the excel file, ***'CommonFeature.xlsx'*** by using openpyxl package of python. Both in DictionaryBuilding and FindingPath, Numpy package of python [17, 18] has been used for rounding off values and Matplotlib package of python [19] for sub-plotting four quadrants of the given image in one single figure (Fig. 3). Data structures like List and Dictionaries of python are used for holding multiple values. List is a data structure which behaves as a dynamic array in python and multiple values can be appended to it [20, 21]. Dictionaries consist of key values and for each key value there will be a specific value [20, 21]. The key values and values for each key value in dictionaries can be a number and can also be a string.

In other words, the proposed system concentrates on dictionary building by extracting features from the images present in ***'Dictionary'*** directory and storing the extracted features in an excel file ***'DictionaryFeatures.xlsx'***. As per the research, the same character in different font sizes results in a number of features. Therefore, it is needed to find out a common feature among all the font sizes. To achieve this ***Longest Common Subsequence (LCS)*** has been used so that there will be one common feature for a particular character. This common feature for the particular character has been stored in an excel file, ***'CommonFeature.xlsx'***. The above process is done by using phases of ***'DictionaryBuilding'***. Then feature is extracted from an input image and this feature is searched in 'CommonFeature.xlsx' by following the phases of ***'FindingMatch'*** to get a correct match. The proposed approach will help to recognise Odia characters from a scanned image or a document converted to image and these recognised characters can be written into a document and further editing can be done.

## II. Related Work

The system introduced in [22], segments handwritten text into lines, from lines, words were segmented and from words characters were segmented. This system had used the water reservoir principle introduced in [23]. The input to the system was a document which was handwritten in Odia. To segment lines, the document was divided to find vertical stripes. Based on vertical projection profile and structural features of Odia characters, text lines were segmented into words. For character segmentation, at first, characters that were connected were detected. Using water-reservoir-concept touching characters of the word were then segmented. The word segmentation module was tested on 3700 words and it was noticed that the word segmentation module had an accuracy of 98.2%. The proposed technique for the isolated and connected character identification had an average accuracy of 96.7%. From the experiment it had been noticed that, in 98.6% cases, isolated characters fall into isolated group. From the experiment it had been noticed that 96.7% accuracy was obtained from two-character touching components. The accuracy of the proposed scheme on three character touching components was 95.1%.

The system introduced in [23] uses a technique for automatic segmentation of handwritten connected numerals. This system had worked on the images of French bank checks from French Company (Itesoft). Initially, the images were in gray scale (256 levels) and they had used histogram based thresholding approach to convert the gray image into binary image. Features were extracted by using the technique called water reservoir. Reservoir was obtained by the accumulation of water poured from the top or from 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: number of reservoirs, position of reservoirs with respect to bounding box of the touching pattern, shape and size of the reservoirs, centre of gravity of the reservoirs and relative positions of the reservoirs. The segmentation result was verified manually and observed that 94.8% of the connected numerals were accurately segmented.

The system described in [24] recognises odia compound character by analysing strokes. The approach had identified 12 strokes that are enough to describe any Odia character. The input character was resized into a 60 x 60 image and then divided into nine equal halves called zones. Each zone consists of some strokes. There are nine zones and 12 strokes so; each feature vector of the character was represented in a 1 x 108. The value of similarity between strokes and zone were arranged in a vector format. Structural Similarity Index had been used as it is based on the concept that the structure of the image is independent of the illumination. The training set had been prepared from the 211 classes of Kalinga font. The system was implemented in windows machine and on MATLAB platform. The independent character recognition accuracy was achieved as 92%. The system also covers many test samples of degraded Kalinga characters. A complete OCR was also designed to work on scanned text document.

The approach described in [25] deals with handwritten Odia character recognition. This system has two level of classification. The input to the first level of classification was a cropped image. Then the input image was binarized followed by thinning. The mid value of the image was found. Then the image was 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 average distance was calculated for that zone. The angle between 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 second-level classification, the cropped image was taken as input and it was divided into nine zones. Then the same procedure that was carried out in first-level classification was also followed in second-level classification. The first-level classification output six average distances and six average angles. The second-level classification also output nine standard deviations, nine average distances and nine average angles. Then Artificial Neural network was used for classification.

The system introduced in [26] considered each character as composition of sequence of high-level strokes and low-level strokes. They had identified low-level strokes in the system explained in [27]. In [26], they had identified forty eight visually non-redundant high-level strokes which form the maximum of a Gujarati character. Each high-level stroke is a combination of point, curves and lines. The proposed method start scanning from the center region of the character in left to right order and extract all junction points. The 3 x 3 neighbourhood 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 endpoint or until next junction point is not reached using contour tracing method. The system had used finite state machine to identify high-level stroke. For classification, the system had used Naive Bayes Classifier and Hidden Markov Model. The overall accuracy achieved using Naive Bayes Classifier and Hidden Markov Model was 93.26% and 96.87% respectively.

## III. System Architecture

This approach consists of ***'DictionaryBuilding'*** and ***'FindingMatch'*** parts. The output of the above two parts are given as input to the Recognition module to find a correct match. The overall system architecture has been shown in Fig. 2

### A. *DictionaryBuilding*

This part deals with building dictionary of features extracted from the dictionary of images of Odia alphabets which are created by scanning a document or a document converted to image by using software, both written in a font family, ***'AkrutiOriAshok-99'*** in a particular font size. The different font sizes used are 18, 20, 22, 24, 26, 28, 32, 48 and 72. For a particular font size, images of Odia alphabets of that font size are stored in a directory. Hence, nine directories are created as nine different font sizes have been used. These nine directories are stored in a directory named as ***'Dictionary'***.

The input to the ***'DictionaryBuilding'*** is the ***'Dictionary'*** directory. The directories in ***'Dictionary'*** and the image files in each directory are accessed using os package of Python. Each image file goes through ***'Feature Extraction in DictionaryBuilding'***.

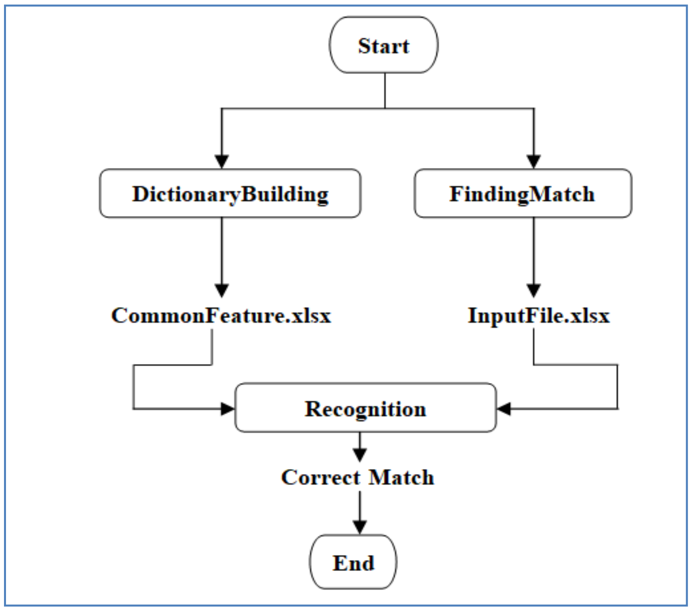

Fig. 2. System Architecture.

#### 1) *Feature Extraction in DictionaryBuilding*

The ***'Dictionary'*** directory consists of nine directories, each directory dedicated to a particular font size. For example, the directory dedicated to font size 18 consists of images of each Odia alphabet written in font size 18, directory dedicated to font size 20 consist of images of each Odia alphabet written in font size 20 and so on. All the images in all these directories of ***'Dictionary'*** undergoes Preprocessing, FindPath, GettingFeaturesRight or GettingFeaturesLeft, RemaingSubQuad, VisitSubQuad modules to extract the features of the images and these extracted features are written into an excel file using WriteToExcel Module. For each directory in the ***'Dictionary'***, a sheet is created in the excel file named as ***'DictionaryFeatures.xlsx'*** and the features are written in that sheet. The overall process of feature extraction of ***'Dictionary'*** images has been shown in Fig. 5.

*a) Preprocessing Module*

The input to the Preprocessing module is the directory ***'Dictionary'.***

***Algorithm:***

Input: Directory ***'Dictionary'***

For each image in the directories of the ***'Dictionary',*** the following steps have been followed:

1. The image is converted to gray image.
2. The white spaces that surround the text in the gray image are removed using Phase – I of RemoveNoise module of [3], that is, RemoveBoundarySpaces. This gives an image that consists of Odia alphabet only.
3. After white spaces have been removed, the image is resized into 64 x 64 by using inter-cubic interpolation.
4. The resized image consisting of Odia alphabet only is then converted into a binary image named as ***'BinaryImage'*** using OSTU's method. In ***'BinaryImage'***, the pixels that form the Odia alphabet are called black pixels and they are represented as 0 whereas the pixels that form the other areas of the ***'BinaryImage'*** are called white pixels and they are represented as 1.
5. The ***'BinaryImage'*** is divided into two equal parts, both horizontally and vertically. In this way, this image is divided into four equal quadrants. The dimension of this image is m X n (m = 64 and n = 64), where ***'m'*** is the number of rows and ***'n'*** is the number of columns. The row that equally divides the ***'BinaryImage'*** horizontally is named as ***'MidRow'*** and it is found out by using the following formula:

   $$MidRow = \lceil m/2 \rceil$$

   The column that equally divides the ***'BinaryImage'*** vertically is named as ***'MidCol'*** and it is found out by using the following formula:

   $$MidCol = \lceil n/2 \rceil$$

6. The four quadrants are found out from the ***'BinaryImage'*** by using ***'MidRow'*** *and* ***'MidCol'***. The four quadrants 1st, 2nd, 3rd and 4th are named as B, C, D and E respectively. The four quadrants are shown in Fig. 3.

   ***B = BinaryImage[0 : MidRow-1, 0 : MidCol-1]***
   ***C = BinaryImage[0 : MidRow-1, MidCol : n]***
   ***D = BinaryImage[MidRow : m, 0 : MidCol-1]***
   ***E = BinaryImage[MidRow : m, MidCol : n]***

7. Call ***FindPath(quadNo, quadrant, DicItem, DicInnerItem, DataPath, shortName)*** for the quadrants B, C, D and E where,

   ***quadNo*** is the number of quadrant among the four quadrants. Here, quadNo = 1, 2, 3, 4
   ***quadrant*** can be B, C, D and E
   ***DicItem*** is the ***d***th directory in the directory ***'Dictionary'***. DicItem = 1, 2, 3, 4, 5, 6, 7, 8 and 9. For each value of DicItem, a sheet in the excel file named as ***'DictionaryFeatures.xlsx'*** is created named with the value of DicItem. For example, if DicItem = 1 then a sheet named '1' is created in the excel file.
   ***DicInnerItem*** is the ***i***th item of the ***DicItem***th directory of ***'Dictionary'***. Each ***'DicInnerItem'*** is an image file. DicInnerItem = 1, 2, 3,........., num where ***'num'*** is the total number of image files in the ***DicItem***th directory.
   ***DataPath*** is the absolute path of the excel file, ***'DictionaryFeatures.xlsx',*** where the features are being written.
   ***shortName*** is the name of the image file present in any ***DicItem***th directory of ***'Dictionary'***.

Suppose quadNo = 1, quadrant = B, DicItem = 2, DicInnerItem = 12 then a sheet named ***'2'*** will be created in the excel file, ***'DictionaryFeatures.xlsx'*** whose path has been provided in the ***'DataPath'*** parameter, and then the extracted feature is being written in the '***12***th' row (as DicInnerItem = 12) and ***'1***st' column (as quadNo = 1) of the sheet. The value in the parameter ***'shortName'*** is written in the fifth column.

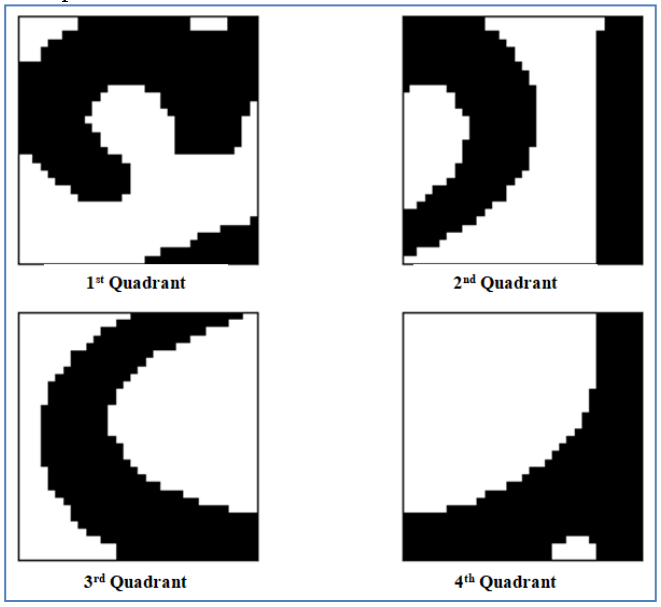

Fig. 3. An Odia Alphabet Divided into Four Quadrants.

*b) FindPath Module*

The steps of FindPath Module are performed for each of the quadrants B, C, D and E. The idea of this module is that the scanning of each quadrant is started from a particular corner and also scanned in a particular direction to extract the features. The scanning of Quadrant B (quadNo = 1) is started from the leftmost and bottom-most corner and when the first black pixel is found, the 'I' and 'J' (co-ordinates of the first black pixel) values are passed to GettingFeaturesRight module for scanning towards right. The scanning of Quadrant C (quadNo = 2) is started from the topmost and leftmost corner and it is scanned towards right using GettingFeaturesRight module. The scanning of Quadrant D (quadNo = 3) is started from the bottom-most and right-most corner and it is scanned towards left using GettingFeaturesLeft module. The scanning of Quadrant E (quadNo = 4) is started from the bottom-most and left-most corner and it is scanned towards right using GettingFeaturesRight module.

***row*** = Number of rows of quadrant

***col*** = Number of columns of quadrant

***Algorithm:***

```
FindPath(quadNo, quadrant, DicItem, DicInnerItem, DataPath, shortName)
1.  SET I = row – 1
2.  SET J = 0
3.  IF quadNo = 1 THEN GO TO STEP 4
4.      REPEAT STEP 5 WHILE J < col
5.        REPEAT STEP 6 WHILE I > 0
6.          IF quadrant[I][J] = 0 THEN GO TO STEP 7
7.              CALL GettingFeaturesRight(I, J, quadrant,
                quadNo, DicItem, DicInnerItem, DataPath,
                shortName)
8.  SET I = 0
9.  SET J = 0
10. IF quadNo = 2 THEN GO TO STEP 11
11.     REPEAT STEP 12 WHILE I < row
12.       REPEAT STEP 13 WHILE J < col
13.         IF quadrant[I][J] = 0 THEN GO TO STEP 14
14.             CALL GettingFeaturesRight(I, J, quadrant,
                quadNo, DicItem, DicInnerItem, DataPath,
                shortName)
15. SET I = row – 1
16. SET J = col – 1
17. IF quadNo = 3 THEN GO TO STEP 18
18.     REPEAT STEP 19 WHILE I > 0
19.       REPEAT STEP 20 WHILE J > 0
20.         IF quadrant[I][J] = 0 THEN GO TO STEP 21
21.           CALL GettingFeaturesLeft(I, J, quadrant,
              quadNo, DicItem, DicInnerItem, DataPath,
              shortName)
22. SET I = row – 1
23. SET J = 0
24. IF quadNo = 4 THEN GO TO STEP 25
25.   REPEAT STEP 26 WHILE I > 0
26.     REPEAT STEP 27 WHILE J < col
27.       IF quadrant[I][J] = 0 THEN GO TO STEP 28
28.           CALL GettingFeaturesRight(I, J, quadrant,
              quadNo, DicItem, DicInnerItem, DataPath,
              shortName)
29. EXIT
```

*c) GettingFeaturesLeft Module*

When the first black pixel is found in FindPath Module while scanning the quadrant from the specified corner, the coordinates of the pixel (I value and J value) are passed to this module to get a continuous trace of black pixels in the specified quadrant. This module scans the quadrant towards the left starting from the first black pixel. This module is used in quadrant D.

***'quadrant'***, ***'quadNo'***, ***'DicItem'***, ***'DicInnerItem'***, ***'DataPath'*** and ***'shortName'*** are explained in the step 7 of preprocessing module.

***'LSubQuad'*** will contain the final feature extracted from a particular quadrant.

***'I'*** and ***'J'*** consists of the row and column number of the first black pixel obtained in a particular quadrant using FindPath module.

***row1*** = Number of rows of quadrant

***col1*** = Number of columns of quadrant

***Algorithm:***

```
GettingFeaturesLeft(I, J, quadrant, quadNo, DicItem, DicInnerItem, DataPath, shortName)
1.  IF J = 0 THEN DO STEPS FROM 2 TO 5
2.    LSubQuad = CALL VisitSubQuad(I, J, 0, row1, 0,
      col1)
3.    CALL RemainingSubQuad(quadrant)
4.    CALL WriteToExcel(DicItem, DicInnerItem, quadNo,
      LSubQuad, DataPath, shortName)
5.    RETURN
6.  ELSE DO STEPS 7 OR 12 OR 17 OR 22,
    WHICHEVER SATISFIES CONDITION FIRST
7.      IF quadrant[I – 1, J - 1] = 0 THEN DO STEPS
        FROM 8 TO 11
8.              I = I – 1
9.              J = J – 1
10.             LSubQuad = CALL VisitSubQuad(I, J, 0,
                row1, 0, col1)
11.             CALL GettingFeaturesLeft(I, J, quadrant,
                quadNo, DicItem, DicInnerItem, DataPath,
                shortName)
12.     ELSE IF quadrant[I, J - 1] = 0 THEN DO STEPS
        FROM 13 TO 16
13.             I = I
14.             J = J – 1
15.             LSubQuad = CALL VisitSubQuad(I, J, 0,
                row1, 0, col1)
16.             CALL GettingFeaturesLeft (I, J, quadrant,
                quadNo, DicItem, DicInnerItem, DataPath,
                shortName)
17.       ELSE IF quadrant[I + 1, J - 1] = 0 THEN DO
          STEPS FROM 18 TO 21
18.             I = I + 1
19.             J = J – 1
20.             LSubQuad = CALL VisitSubQuad(I, J, 0,
                row1, 0, col1)
21.             CALL GettingFeaturesLeft (I, J, quadrant,
                quadNo, DicItem, DicInnerItem, DataPath,
                shortName)
22.       ELSE IF (quadrant[I – 1, J - 1] = 1 AND
          quadrant[I, J - 1] = 1 AND quadrant[I + 1, J - 1] =
          1) OR (J = 0) OR (I = 0) OR (I = row1 – 1) OR (J =
          col1 – 1) THEN DO STEPS FROM 23 TO 25
```

```
23.             CALL RemainingSubQuad(quadrant)
24.             CALL WriteToExcel(DicItem, DicInnerItem,
                quadNo, LSubQuad, DataPath, shortName)
25.             RETURN
26.     EXIT
```

*d) GettingFeaturesRight Module*

The aim of this module is to get a continuous trace of black pixels scanning from left to right. When the first black pixel is found in FindPath Module while scanning the quadrant from the specific corner, the coordinates of the pixel (I value and J value) are passed to this module to get a continuous trace of black pixels in the specified quadrant. This module is used in quadrants B, C and E.

***'quadrant'***, ***'quadNo'***, ***'DicItem'***, ***'DicInnerItem'***, ***'DataPath'*** and ***'shortName'*** are explained in the step 7 of Preprocessing module.

***'LSubQuad'*** will contain the final feature extracted from a particular quadrant.

***'I'*** and ***'J'*** consists of the row and column number of the first black pixel obtained in a particular quadrant using FindPath module.

***row1*** = Number of rows of quadrant

***col1*** = Number of columns of quadrant

***Algorithm:***

```
GettingFeaturesRight(I, J, quadrant, quadNo, DicItem,
DicInnerItem, DataPath, shortName)
1.  IF J = col1 - 1 THEN DO STEPS FROM 2 TO 5
2.      LSubQuad = CALL VisitSubQuad(I, J, 0, row1, 0,
        col1)
3.      CALL RemainingSubQuad(quadrant)
4.      CALL WriteToExcel(DicItem, DicInnerItem,
        quadNo, LSubQuad, DataPath, shortName)
5.      RETURN
6.  ELSE DO STEPS 7 OR 12 OR 17 OR 22 WHICHEVER
    CONDITION SATISFIES FIRST
7.      IF quadrant[I – 1, J + 1] = 0 THEN DO STEPS
        FROM 8 TO 11
8.          I = I – 1
9.          J = J + 1
10.         LSubQuad = CALL VisitSubQuad(I, J, 0,
            row1, 0, col1)
11.         CALL GettingFeaturesRight(I, J, quadrant,
            quadNo, DicItem, DicInnerItem, DataPath,
            shortName)
12.     ELSE IF quadrant[I, J + 1] = 0 THEN DO STEPS
        FROM 13 TO 16
13.         I = I
14.         J = J + 1
15.         LSubQuad = CALL VisitSubQuad(I, J, 0,
            row1, 0, col1)
16.         CALL GettingFeaturesRight(I, J, quadrant,
            quadNo, DicItem, DicInnerItem, DataPath,
            shortName)
17.     ELSE IF quadrant[I + 1, J + 1] = 0 THEN DO
        STEPS FROM 18 TO 21
18.         I = I + 1
19.         J = J + 1
20.         LSubQuad = CALL VisitSubQuad(I, J, 0,
            row1, 0, col1)
21.         CALL GettingFeaturesRight(I, J, quadrant,
            quadNo, DicItem, DicInnerItem, DataPath,
            shortName)
22.     ELSE IF (quadrant[I – 1, J + 1] = 1 AND
        quadrant[I, J + 1] = 1 AND quadrant[I + 1, J + 1]
        = 1) OR (J = 0) OR (I = 0) OR (I = row1 – 1) OR
        (J = col1 – 1) THEN DO STEPS FROM 23 TO
        25
23.         CALL RemainingSubQuad(quadrant)
24.         CALL WriteToExcel(DicItem,
            DicInnerItem, quadNo, LSubQuad,
            DataPath, shortName)
25.         RETURN
26. EXIT
```

*e) VisitSubQuad Module*

Each of the four quadrants **B, C, D** and **E** is again divided into four sub-quadrants named as a, b, c and d. This module uses a list data structure named as ***'subQuad'*** and appends name of the sub-quadrant (***'a'*** or ***'b'*** or ***'c'*** or ***'d'***) for each black pixel found while scanning by GettingFeaturesLeft or GettingFeaturesRight module including the first black pixel found in the FindPath Module in ***'SubQuad'***. The sub-quadrants of each quadrant have been shown in Fig. 4. The value returned by this module is stored in ***'LSubQuad'*** of either GettingFeaturesLeft or GettingFeaturesRight module. The final value in ***'LsubQuad'*** is the feature extracted for a particular quadrant (***'B'*** or ***'C'*** or ***'D'*** or ***'E'***).

***'srow'*** is the row number where the quadrant starts.

***'erow'*** is the row number where the quadrant ends.

***'scol'*** is the column number where the quadrant starts.

***'ecol'*** is the column number where the quadrant ends.

$$frow = \left[\left((erow - srow)\right)/2\right]$$

$$fcol = \left[\left((ecol - scol)\right)/2\right]$$

***Algorithm:***

```
VisitSubQuad(I, J, srow, erow, scol, ecol)
1.  IF (I >= srow AND I <= frow – 1) AND (J >= scol AND
    J <= fcol – 1) GO TO STEP 2
2.      APPEND 'a' in 'subQuad'
3.  ELSE IF (I >= srow AND I <= frow – 1) AND (J >= fcol
    AND J <= ecol) GO TO STEP 4
4.      APPEND 'b' in 'subQuad'
5.  ELSE IF (I >= frow AND I <= erow) AND (J >= fcol
    AND J <= ecol) GO TO STEP 6
6.      APPEND 'c' in 'subQuad'
7.  ELSE IF (I >= frow AND I <= erow) AND (J >= scol
    AND J <= fcol - 1) GO TO STEP 8
8.      APPEND 'd' in 'subQuad'
9.  RETURN 'subQuad'
10. EXIT
```

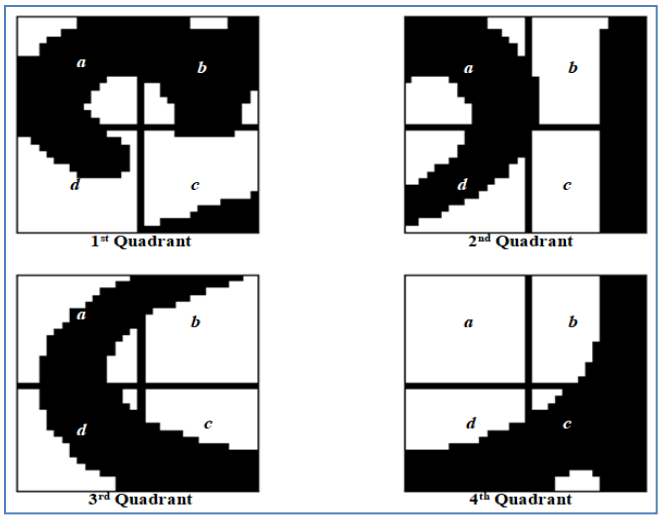

Fig. 4. Sub-Quadrants of Four Quadrants.

### *f) RemainingSubQuad Module*

A continuous trace of black pixels is found by either GettingFeaturesLeft or GettingFeaturesRight starting from the first black pixel found in FindPath module and the name of the sub-quadrant is appended in ***'LSubQuad'***. But the continuous trace of black pixels may not have accessed some part of the quadrant (***'B'*** or ***'C'*** or ***'D'*** or ***'E'***). To ensure, all parts of the quadrant have been accessed, the remaining parts are accessed using RemainingSubQuad Module. First, this module checks if all the sub-quadrants have been accessed for the black pixels. This is done by checking the contents of the ***'LSubQuad'*** list. In other words, if a sub-quadrant does not have any black pixel then that sub-quadrant is not allowed to be present in the ***'LSubQuad'*** list. If the name of the all sub-quadrants that have black pixels have appeared at least once in ***'LSubQuad'*** then, RemainingSubQuad exits, otherwise, RemainingSubQuad is called recursively to scan the sub-quadrants until all sub-quadrants that have black pixels have been scanned and stored in the ***'LSubQuad'***.

***'quadrant'*** consists of ***'B'***, ***'C'***, ***'D'*** and ***'E'***.

***Algorithm:***

```
RemainingSubQuad(quadrant)
1.  IF 'a', 'b', 'c' and 'd' all are in 'LSubQuad' THEN
2.      GO TO STEP 20
3.  ELSE
4.      IF 'a' IS NOT IN 'LSubQuad' THEN DO STEP 5 TO 7
5.          SCAN black pixels of quadrant from top-most and left-most corner to find the first black pixel and from there scan towards right following the similar procedure as in GettingFeaturesRight to find the continuous trace of black pixel.
6.          For each black pixel in the continuous trace APPEND 'a' in 'LSubQuad'.
7.          CALL RemainingSubQuad(quadrant)
8.      ELSE IF 'b' IS NOT IN 'LSubQuad' THEN DO STEP 9 TO 11
9.          SCAN black pixels of quadrant from top-most and right-most corner to find the first black pixel and from there scan towards left following the similar procedure as in GettingFeaturesLeft to find the continuous trace of black pixel.
10.         For each black pixel in the continuous trace APPEND 'b' in 'LSubQuad'.
11.         CALL RemainingSubQuad(quadrant)
12.     ELSE IF 'c' IS NOT IN 'LSubQuad' THEN DO STEP 13 TO 15
13.         SCAN black pixels of quadrant from bottom-most and right-most corner to find the first black pixel and from there scan towards left following the similar procedure as in GettingFeaturesLeft to find the continuous trace of black pixel.
14.         For each black pixel in the continuous trace APPEND 'c' in 'LSubQuad'.
15.         CALL RemainingSubQuad(quadrant)
16.     ELSE IF 'd' IS NOT IN 'LSubQuad' THEN DO STEP 17 TO 19
17.         SCAN black pixels of quadrant from bottom-most and left-most corner to find the first black pixel and from there scan towards right following the similar procedure as in GettingFeaturesRight to find the continuous trace of black pixel.
18.         For each black pixel in the continuous trace APPEND 'd' in 'LSubQuad'.
19.         CALL RemainingSubQuad(quadrant)
20. EXIT
```

### *g) WriteToExcel Module*

When a quadrant (***'B'*** or ***'C'*** or ***'D'*** or ***'E'***) is scanned completely for tracing black pixels, all the features of the quadrant are extracted in the form of a, b, c and d and stored in ***'LSubQuad'***. The contents of the ***'LSubQuad'*** are concatenated to present the features in a string format. This string value is written in the ***'DicItem***$^{th}$***'*** sheet, ***'DicInnerItem***$^{th}$***'*** row and ***'quadNo***$^{th}$***'*** column of the excel file, ***'DictionaryFeatures.xlsx'***. The path of the excel file is stored in ***'DataPath'*** parameter. The ***'shortName'*** is the file name and it is written in the fifth column and ***'DicInnerItem***$^{th}$***'*** row of the ***'DicItem***$^{th}$***'*** sheet of the excel file. All the extracted features are written in the excel file by using the openpyxl package of Python. This excel file contains the features of each alphabet in all font sizes (18, 20, 22, 24, 26, 28, 32, 48 and 72). For example, the alphabet ઘ in font size 18, 20, 22, 24, 26, 28, 32, 48 and 72 is written in the first row of sheet named 1, 2, 3, 4, 5, 6, 7, 8 and 9 respectively.

***Algorithm:***

```
WriteToExcel(DicItem, DicInnerItem, quadNo, LSubQuad, DataPath, shortName)
1.  INITIALIZE 'S' to an empty string
2.  A = 0
3.  REPEAT STEP 4 WHILE A < LENGTH(LSubQuad)
4.    S = S + LSubQuad[A]
5.  For each value of DicItem, CREATE a new sheet named with the value of the DicItem.
```

```
6.  IF the current value of DicItem is same as the value in the previous iteration THEN a new sheet is not created and the feature in string format is written in the current sheet of the excel file, 'DictionaryFeatures.xlsx'.
7.  ELSE CREATE a new sheet for writing extracted features of the alphabets of next font size in the excel file, 'DictionaryFeatures.xlsx'.
8.  EXIT
```

### *h) CommonFeature Module*

When all the features have been extracted and written in the excel file, ***'DictionaryFeatures.xlsx'*** for all alphabets in all font sizes using WriteToExcel module then, a common feature is found from all the extracted features of an alphabet in different font sizes. For this, CommonFeature module is used. This module finds the ***LCS (Longest Common Subsequence)*** of all the features of a particular alphabet in different font sizes to find the common feature. LCS is a way of finding longest common sub-sequences from a set of sequences.The common feature found using LCS is written in another excel file named as ***'CommonFeature.xlsx'*** which consists of only one sheet.

***'fiQuList'***, ***'SeQuList'***, ***'ThQuList'*** and ***'FoQuList'*** are the lists that contains features of first quadrant (B), second quadrant (C), third quadrant (D) and fourth quadrant (E) of a particular alphabet in different font sizes respectively.

***'row'*** is the total number of rows present in the excel file, ***'DictionaryFeatures.xlsx'*** which contains features of all alphabets in a sheet. The value of ***'row'*** is same in all sheets of the excel file, ***'DictionaryFeatures.xlsx'***.

***'sheet'*** is the total number of sheets present in the excel file, ***'DictionaryFeatures.xlsx'*** which contains features of all alphabets in different font sizes (In this research, sheet = 9 as nine different font sizes are considered).

***'Text1'***, ***'Text2'***, ***'Text3'*** and ***'Text4'*** are strings.

***'sr'*** is initialized to 0.

***'sh'*** is initialized to 0.

***Algorithm:***

```
CommonFeature()
1.  REPEAT STEP 2 WHILE sr < row
2.      REPEAT STEPS FROM 3 TO 6 WHILE sh < sheet
3.          APPEND the feature in first column of 'sr^th' row of 'sh^th' sheet of 'DictionaryFeatures.xlsx' in 'fiQuList'.
4.          APPEND the feature in second column of 'sr^th' row of 'sh^th' sheet of 'DictionaryFeatures.xlsx' in 'SeQuList'.
5.          APPEND the feature in third column of 'sr^th' row of 'sh^th' sheet of 'DictionaryFeatures.xlsx' in 'ThQuList'.
6.          APPEND the feature in fourth column of 'sr^th' row of 'sh^th' sheet of 'DictionaryFeatures.xlsx' in 'FoQuList'.
7.      Text1 = fiQuList[0]
8.      f = 0
9.      REPEAT STEP 10 WHILE f < (LENGTH (fiQuList) – 1)
10.         Text1 = FindLCS(Text1, fiQuList[f + 1]
11.     Text2 = SeQuList[0]
12.     f = 0
13.     REPEAT STEP 14 WHILE f < (LENGTH (SeQuList) – 1)
14.         Text2 = FindLCS(Text2, SeQuList[f + 1]
15.     Text3 = ThQuList[0]
16.     f = 0
17.     REPEAT STEP 18 WHILE f < (LENGTH (ThQuList) – 1)
18.         Text3 = FindLCS (Text3, ThQuList[f + 1])
19.     Text4 = FoQuList[0]
20.     f = 0
21.     REPEAT STEP 22 WHILE f < (LENGTH (FoQuList) – 1)
22.         Text4 = FindLCS (Text4, FoQuList[f + 1])
23.     WRITE the value of 'Text1' in the first column of 'sr^th' row of the excel file, 'CommonFeature.xlsx'.
24.     WRITE the value of 'Text2' in the second column of 'sr^th' row of the excel file, 'CommonFeature.xlsx'.
25.     WRITE the value of 'Text3' in the third column of 'sr^th' row of the excel file, 'CommonFeature.xlsx'.
26.     WRITE the value of 'Text4' in the fourth column of 'sr^th' row of the excel file, 'CommonFeature.xlsx'.
27.     Clear all the lists 'FiQuList', 'SeQuList', 'ThQuList' and 'FoQuList'.
28.     INITIALIZE 'Text1', 'Text2', 'Text3' and 'Text4' to empty string.
29. EXIT
```

### *i) Longest Common Subsequence*

This module finds and returns the LCS (Longest Common Subsequence) of ***'String1'*** and ***'String2'***. The LCS algorithm has been implemented using Dynamic Programming in this paper. If ***'String1'*** and ***'String2'*** are equal then ***'String1'*** is stored in ***'ls'*** and it is returned, otherwise the LCS of ***'String1'*** and ***'String2'*** is found out and it is stored in ***'revLs'*** and returned. The two arrays ***'LcsForm'*** and ***'b'*** are used to store the length of the LCS and the traversing direction of the LCS respectively in each column of each row. The ***'s'***, ***'u'*** and ***'l'*** denote ***'towards diagonal'***, ***'towards upper'*** and ***'towards left'*** directions respectively. After all the values of ***'LcsForm'*** and ***'b'*** are found out, both arrays are scanned from the bottom-most corner and right-most side to get the value of **I** and **J** where ***LcsForm[I][J] = MaxValue and b[I][J] = 's'*** and for each ***'s'*** in ***'b'*** array, the common item in both the strings (String1 and String2) is appended in ***'ls'***. The ***'MaxValue'*** is the maximum length of LCS in '***LcsForm'***. At last ***'ls'*** is reversed and the result is stored in ***'revLs'***.

***Algorithm:***

```
FindLCS(String1, String2)
1.  IF String1 = String2 THEN DO STEP 2 TO 3
2.      APPEND 'String1' in the list named as 'ls'.
3.      RETURN 'ls'.
4.  ELSE DO FROM STEP 5 TO 37
5.      m = LENGTH(String1)
```

```
6.    n = LENGTH(String2)
7.    INITIALIZE the array 'LcsForm' with dimensions
      (m + 1, n + 1) to zero.
8.    INITIALIZE the array 'b' with dimensions (m + 1, n
      + 1) to zero.
9.    I = 0
10.   J = 0
11.   REPEAT STEP 12 WHILE I < (m + 1)
12.     REPEAT STEP 13 or 16 or 19 WHICEVER
        SATISFIES THE CONDITION FIRST WHILE J <
        (n + 1)
13.       IF String1[I – 1] = String2[J – 1] THEN DO
          STEP 14 TO 15
14.           LcsForm[I][J] = LcsForm[I][J] + 1
15.           b[I][J] = 's'
16.       ELSE IF LcsForm[I – 1][J] >= LcsForm[I][J – 1]
          THEN DO STEP 17 TO 18
17.           LcsForm[I][J] = LcsForm[I – 1][J]
18.           b[I][J] = 'u'
19.         ELSE DO STEP 20 TO 21
20.           LcsForm[I][J] = LcsForm[I][J – 1]
21.           b[I][J] = 'l'
22.   Find the maximum value in the array 'LcsForm' and
      it is stored in 'MaxValue'.
23.   Search the array 'LcsForm' from right-most side and
      bottom-most corner of the array and find the value of
      I and J in the array where LcsForm[I][J] = MaxValue
      and b[I][J] = 's'.
24.   After values of I and J are found for LcsForm[I][J] =
      MaxValue and b[I][J] = 's', DO STEP 25
25.     REPEAT STEP 26 or 30 or 33 WHICHEVER
        SATISFIES THE CONDITION FIRST WHILE I >
        0 AND J > 0
26.       IF b[I][J] = 's' THEN DO STEP 27 TO 29
27.           APPEND the value in String1[I][J] in the list
              'ls'.
28.           I = I – 1
29.           J = J – 1
30.       ELSE IF b[I][J] = 'u' DO STEP 31 TO 32
31.           I = I – 1
32.           J = J
33.       ELSE IF b[I][J] = 'l' DO STEP 34 TO 35
34.           I = I
35.           J = J – 1
36.   REVERSE the items of the list 'ls' and store it in
      'revLs'.
37.   RETURN 'revLs'.
38. EXIT
```

Hence, the final output of the ***'DictionaryBuilding'*** is the ***'CommonFeature.xlsx'*** excel file which consists of the common feature for each alphabet extracted from features present in ***'DictionaryFeatures.xlsx'*.** For example, the final common features for the alphabet are:

1st quadrant – aaaaaaaaaaaaabbbbbbcccccccccccccc

2nd quadrant – aaaaaaaabbbbbccccccdddddddddddd

3rd quadrant – cccccccccccccccdddddaaaabbbbbbbbbbbbaaaaaaaaa

4th quadrant - dddddddddddddddcccccbbbbbbbbbb

## *B. FindingMatch*

This part deals with finding a correct match from the dictionary of common features stored in the excel file, ***'CommonFeature.xlsx'*** when an image of Odia alphabet is provided as input. This input image is stored in a directory named as ***'Input'***. The ***'FindingMatch'*** part undergoes through two phases: ***'Feature Extraction'*** and ***'Recognition'***.

### *1) Feature Extraction*

This phase undergoes through seven modules for extracting features from the input image present in the directory ***'Input'*** and the features are written to an excel file named as ***'InputFile.xlsx'***. The different modules are: Preprocessing, FindPath, GettingFeaturesRight or GettingFeaturesLeft, VisitSubQuad and RemainingSubQuad for extracting features from the input image and the features are written in the excel file using WriteToExcel Module. The overall process of feature extraction of Input image has been shown in Fig. 5.

#### *a) Preprocessing Module*

The steps in this module are same as described in Preprocessing module of ***'DictionaryBuilding'*** except the values passed to the parameters in FindPath module. The input to this module is the directory ***'Input'*** consisting of an image of Odia alphabet. The input image is converted to gray image. The white spaces surrounding the Odia alphabet in the gray image are removed using the Phase – 1 of RemoveNoise module of [3] (RemoveBoundarySpaces). Then the resultant image is resized into the dimension p x q (p = 64 and q = 64) where, ***'p'*** is the number of rows and ***'q'*** is the number of columns**.** Then the resized image is converted to binary image named as ***'BinayImageIn'.*** The row that equally divides the ***'BinayImageIn'*** horizontally is named as ***'MidRow'*** and it is found out by using the following formula:

$$MidRow = \left[{}^{p}/_{2}\right]$$

The column that equally divides the ***'BinayImageIn'*** vertically is named as ***'MidCol'*** and it is found out by using the following formula:

$$MidCol = \left[{}^{q}/_{2}\right]$$

The four quadrants are found out from ***'BinayImageIn'*** in the following way:

***U = BinaryImageIn[0 : MidRow-1, 0 : MidCol-1]***

***V = BinaryImageIn[0 : MidRow-1, MidCol : n]***

***W = BinaryImageIn[MidRow : m, 0 : MidCol-1]***

***X = BinaryImageIn[MidRow : m, MidCol : n]***

***'U'***, ***'V'***, ***'W'*** and ***'X'*** are the first, second, third and fourth quadrant respectively.

Then CALL FindPath(quadNo, quadrant, DicItem, DicInnerItem, DataPath, shortName) for the quadrants U, V, W and X.

*b) FindPath Module*

The steps in this module are same as described in FindPath module of ***'DictionaryBuilding'*** except that the steps of FindPath Module are performed for each of the quadrants U, V, W and X. Here ***'DicItem'*** and ***'DicInnerItem'*** are constants and are set to 1 as the ***'Input'*** folder has no sub-directories and it has only one image at any given time. The ***'DataPath'*** parameter holds the absolute path of the excel file named as ***'InputFile.xlsx'*** and in this file features of all the four quadrants of the input image are being written. The features of ***'U'***, ***'V'***, ***'W'*** and ***'X'*** are written in first, second, third and fourth column of ***'InputFile.xlsx'*** respectively.

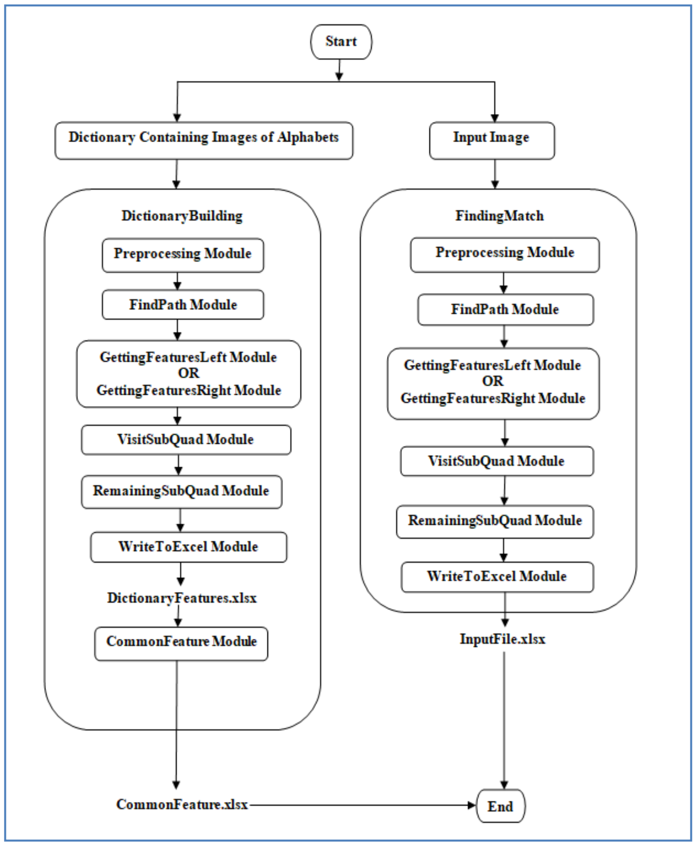

Fig. 5. Feature Extraction for Dictionary of Images and Input Image.

*c) GettingFeaturesLeft Module*

The steps in this module are same as described in GettingFeaturesLeft module of ***'DictionaryBuilding'*** except that the steps here are applied to W quadrant.

*d) GettingFeaturesRight Module*

The steps in this module are same as described in GettingFeaturesRight module of ***'DictionaryBuilding'*** except that the steps here are applied to U, V and X quadrants.

*e) VisitSubQuad Module*

The steps in this module are same as described in VisitSubQuad module of ***'DictionaryBuilding'*** except that the steps are applied to U, V, W and X quadrants. Similar to as explained in the VisitSubQuad module of ***'DictionaryBuilding'***, the quadrants are divided into four sub-quadrants, a, b, c and d. For each black pixel in the continuous trace, the sub-quadrant (either ***'a'*** or ***'b'*** or ***'c'*** or ***'d'***) is found out and appended in ***'subQuad'***. The value of ***'subQuad'*** is returned and set to ***'LSubQuad'*** in ***'GettingFeaturesLeft'*** or ***'GettingFeaturesRight'***, whichever has been called.

*f) RemainingSubQuad Module*

The steps in this module are same as described in RemainingSubQuad module of ***'DictionaryBuilding'*** except that the steps are applied to U, V, W and X quadrants. If any portions of the quadrants U, V, W and X are not covered by the continuous trace of black pixels, those remaining portions are covered by this module and the name of sub-quadrants (either ***'a'*** or ***'b'*** or ***'c'*** or ***'d'***) are appended in ***'LSubQuad'***.

*g) WriteToExcel Module*

The steps in this module are same as described in WriteToExcel module of ***'DictionaryBuilding'*** except that the features extracted from the quadrants U, V, W and X are written in an excel file named as ***'InputFile.xlsx'***. The absolute path of ***'InputFile.xlsx'*** is stored in the ***'DataPath'*** parameter and the file name of input image is stored in ***'shortName'***. The value in ***'shortName'*** parameter is written in the fifth column of ***'InputFile.xlsx'.*** Hence, the feature extracted from the quadrants U, V, W, X and the value in ***'shortName'*** parameter are written in the first, second, third, fourth and fifth column of the first row of the excel file, ***'InputFile.xlsx'*** respectively and there is only one sheet present in the excel file as there is no sub-directories of the ***'Input'*** directory. For example, the final feature for the input image are:

1st quadrant – aaaaaaaaaaaaaaabbbbbbbbbbbbbbbbbccccccccccccccddd

2nd quadrant – aaaaaaaaaaaaabbbbbbbbccccccccdddddddddddddd

3rd quadrant – ccccccccccccccdddddaaaaaaabbbbbbbbbbbbaaaaaaaaaa

4th quadrant - ddddddddddddddddccccbbbbbbbbbbbb

*2) Recognition*

This phase undergoes through three modules: CheckCommonFeature module, MatchCommonFeature module and TraceAnotherDirection Module. The overall process of recognition has been shown in Fig. 6.

***InCol:*** number of columns in the ***'Input.xlsx'***

***ComRow:*** number of rows in the ***'CommonFeature.xlsx'***

***'InpPat'*** is a list consisting of the final features of 1st, 2nd, 3rd and 4th quadrants for the ***'eth'*** row of ***'Input.xlsx'.***

***'QuList'*** is a list consisting of the final features of 1st, 2nd, 3rd and 4th quadrants for the ***'fth'*** row of ***'CommonFeature.xlsx'.***

***'MatchFirst'*** is a list consisting of the file names of the matched features obtained as the output of CheckCommonFeature module.

***'MatchedRow'*** is a list consisting of the row numbers of the matched features in ***'CommonFeature.xlsx'.***

***'MatchSecond'*** is a list consisting of the output of MatchCommonFeature module.

***Algorithm:***

```
Recognition( )
1.  SET e = 1
2.  REPEAT STEP 3 WHILE e <= InCol
3.      APPEND the feature present in '1st' row and 'eth' column of 'Input.xlsx' in the list 'InpPat'.
4.  SET f = 1
5.  REPEAT STEPS FROM 6 TO 17 WHILE f < ComRow
6.      APPEND the feature present in the 'fth' row and '1st' column of 'CommonFeature.xlsx' in the list 'QuList'.
7.      APPEND the feature present in the 'fth' row and '2nd' column of 'CommonFeature.xlsx' in the list 'QuList'.
8.      APPEND the feature present in the 'fth' row and '3rd' column of 'CommonFeature.xlsx' in the list 'QuList'.
9.      APPEND the feature present in the 'fth' row and '4th' column of 'CommonFeature.xlsx' in the list 'QuList'.
10.     ite = 1
11.     REPEAT STEPS FROM 12 TO 14 WHILE ite <= LENGTH (InpPat)
12.         Param1 = CALL CheckCommonFeature(InpPat[ite], QuList[ite])
13.         IF Param1 = 1 THEN GO TO STEP 14
14.             Param3 = Param3 + 1
15.     IF Param3 = 4 THEN GO TO STEP 16
16.         RETRIEVE the file name of the matched image feature present in the 'fth' row and '5th' column of the 'CommonFeature.xlsx' and APPEND the file name in a list named as 'MatchFirst' and 'fth' row number of the matched image feature in a list 'MatchedRow'.
17.     CLEAR the list 'QuList'.
18. IF LENGTH(MatchFirst) > 1 THEN DO STEPS 19 TO 33
19.     SET f = 1
20.     REPEAT STEPS FROM 21 TO 33 WHILE f <= LENGTH (MatchFirst)
21.         APPEND the feature present in the '(MatchedRow[f])th' row and '1st' column of 'CommonFeature.xlsx' in the list 'QuList'.
22.         APPEND the feature present in the '(MatchedRow[f])th' row and '2nd' column 'CommonFeature.xlsx' in the list 'QuList'.
23.         APPEND the feature present in the '(MatchedRow[f])th' row and '3rd' column 'CommonFeature.xlsx' in the list 'QuList'.
24.         APPEND the feature present in the '(MatchedRow[f])th' row and '4th' column 'CommonFeature.xlsx' in the list 'QuList'.
25.         SET ite = 1
26.         REPEAT STEPS FROM 27 TO 29 WHILE ite <= LENGTH(InpPat)
27.             Param2 = CALL MatchCommonFeature (InpPat[ite], QuList[ite])
28.             IF Param2 = 1 THEN GO TO STEP 29
29.                 Param4 = Param4 + 1
30.         IF Param4 = 4 THEN DO STEPS 31 TO 32
31.             RETRIEVE the file name of the matched image feature present in the '(MatchedRow[f])th' row and '5th' column of the 'CommonFeature.xlsx' and APPEND the file name in a list named as 'MatchSecond'.
32.             PRINT 'MatchSecond'
33.         CLEAR the list 'QuList'
34. ELSE GO TO STEP 35
35.     PRINT the list 'MatchFirst'
36. EXIT
```

*a) CheckCommonFeature Module*

When feature extraction of the input image has been completed, features of all the four quadrants U, V, W and X are written in the excel file named as ***'InputFile.xlsx'***. The common feature of a particular alphabet that is extracted from all the features present in ***'DictionaryFeatures.xlsx'*** file has been written in ***'CommonFeature.xlsx'***. The common feature of first quadrant present in the first column of each row of ***'CommonFeature.xlsx'*** is searched to find whether the sequence of common feature is present in the first column of ***'InputFile.xlsx'***. The same search procedure is repeated for second, third and fourth columns of both the files, ***'CommonFeature.xlsx'*** and ***'InputFile.xlsx'.*** In other words, the second, third and fourth columns of each row of ***'CommonFeature.xlsx'*** are searched in second, third and fourth columns of ***'InputFile.xlsx'*** respectively. For this, the CheckCommonFeature(String1, String2) has been used. The presence of common features of ***'CommonFeature.xlsx'*** in the features of ***'InputFile.xlsx'*** in a continuous form or non-continuous form helps to find a correct match. If the features in the first, second, third and fourth columns of ***'InputFile.xlsx'*** are present in the first, second, third and fourth columns of ***'CommonFeature.xlsx'*** respectively in a particular row then Param1 is set to 1 otherwise it is set to 0. If Param1 = 1 then Param3 is incremented by 1. For example, if the value of Param3 is 4 after all the columns of ***'InputFile.xlsx'*** have been checked with the respective columns of ***'CommonFeature.xlsx'*** for all rows, then the file name is retrieved from the fifth column of the row that consists of matched image feature in ***'CommonFeature.xlsx'.*** The file name from fifth column of matched image feature is appended in the list ***'MatchFirst'*** and the row number of matched image feature is appended in the list ***'MatchedRow'***. In some cases, the list ***'MatchFirst'*** have more than one correct match and in these cases ***'MatchCommonFeature'*** module is called.

*b) MatchCommonFeature Module*

This module is used when the list ***'MatchFirst'*** (output of ***'CheckCommonFeature')*** consists of more than one match. In this module, the common features from the first, second, third and fourth columns of the each row number that is present in the list ***'MatchedRow'*** are retrieved from ***'CommonFeature.xlsx'*** and appended in the list ***'QuList'***. The features present in the first, second, third and fourth columns present in ***'InputFile.xlsx'*** are retrieved and appended in the list '***Inpat'.*** Then the LCS (Longest Common Sequence) of the two strings, ***'Str1'*** and ***'Str2'*** is found where Str1 = QuList[ite] and Str2 = Inpat[ite], ite = 1, 2, 3, 4 and the resultant LCS is matched with ***'Str2'***. If the resultant LCS gets a match with ***'Str2'*** then this module returns 1, otherwise 0. The return value of this module is stored in Param2. If Param2 = 1 then, Param4 is incremented by 1. This process is done for each item present in the list ***'MatchFirst'***. If Param4 = 4 for an item in ***'MatchFirst'*** then that file name is copied to the list ***'MatchSecond'***. According to the research, the ***'MatchCommonFeature'*** selects the correct match from the multiple matches in the list ***'MatchFirst'***. But if for some images, both the modules of ***'Recognition'*** result in multiple matches or no matches, then ***'TraceAnotherDirection'*** module is called.

### *c) TraceAnotherDirection Module*

This module consists of two parts '***DictionaryAnotherWay'*** and ***'FindingMatchAnother'***. For extraction of features from the dictionary of images stored in the directory ***'Dictionary'***, the ***'DictionaryAnotherWay'*** undergoes same modules as in ***'DictionaryBuilding'***, the only difference being the direction of tracing of the continuous black pixels in the '***FindPath'*** module in ***'DictionaryBuilding'*** for the four quadrants. The modified direction of the tracing of the continuous black pixels is shown in a module named ***'FindPathAnother'***.

***row*** = Number of rows of quadrant

***col*** = Number of columns of quadrant

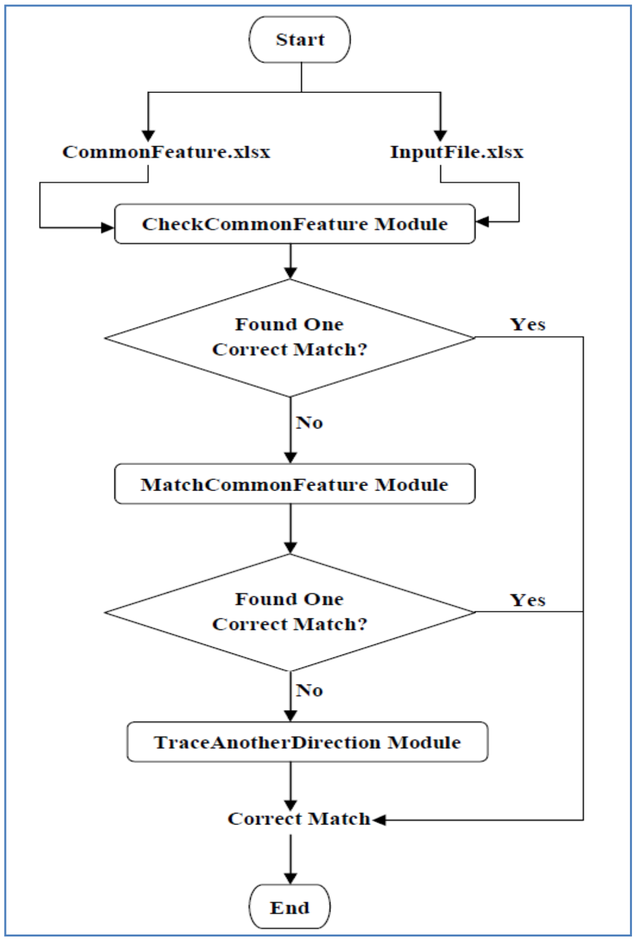

Fig. 6. Recognition of Input Image.

***Algorithm:***

```
FindPathAnother(quadNo, quadrant, DicItem, DicInnerItem, DataPath, shortName)
1.  SET I = 0
2.  SET J = col – 1
3.  IF quadNo = 1 THEN GO TO STEP 4
4.      REPEAT STEP 5 WHILE J > 0
5.          REPEAT STEP 6 WHILE I < row
6.              IF quadrant[I][J] = 0 THEN GO TO STEP 7
7.                  CALL GettingFeaturesLeft(I, J, quadrant, quadNo, DicItem, DicInnerItem, DataPath, shortName)
8.  SET I = row – 1
9.  SET J = col – 1
10. IF quadNo = 2 THEN GO TO STEP 11
11.     REPEAT STEP 12 WHILE I > 0
12.         REPEAT STEP 13 WHILE J > 0
13.             IF quadrant[I][J] = 0 THEN GO TO STEP 14
14.                 CALL GettingFeaturesLeft(I, J, quadrant, quadNo, DicItem, DicInnerItem, DataPath, shortName)
15. SET I = 0
16. SET J = 0
17. IF quadNo = 3 THEN GO TO STEP 18
18.     REPEAT STEP 19 WHILE I < row
19.         REPEAT STEP 20 WHILE J < col
20.             IF quadrant[I][J] = 0 THEN GO TO STEP 21
21.                 CALL GettingFeaturesRight(I, J, quadrant, quadNo, DicItem, DicInnerItem, DataPath, shortName)
22. SET I = 0
23. SET J = col – 1
24. IF quadNo = 4 THEN GO TO STEP 25
25.     REPEAT STEP 26 WHILE I < row
26.         REPEAT STEP 27 WHILE J > 0
27.             IF quadrant[I][J] = 0 THEN GO TO STEP 28
28.                 CALL GettingFeaturesLeft(I, J, quadrant, quadNo, DicItem, DicInnerItem, DataPath, shortName)
29. EXIT
```

***'FindingMatchAnother'*** finds a correct match from the '***CommonFeature2.xlsx'***(output of ***'DictionaryAnotherWay'). ***For Feature extraction, the input image present in the directory ***'Input'*** undergoes same modules as in ***'FindingMatch'***, except the direction of tracing of the continuous black pixels in the '***FindPath'*** module in ***'FindingMatch'*** for the four quadrants. The modified direction of the tracing of the continuous black pixels is shown in a module named ***'FindPathAnother'***. The same modules of ***'Recognition'*** are used for finding a correct match. As per the research, the ***'TraceAnotherDirection'*** gives a correct match for the input image.

## IV. Results

This paper deals with recognising a printed Odia alphabet in an image which is created by scanning a document or document converted to image by using a software, both written in a font family ***'AkritiOriAshok-99'*** in a particular font size. The font sizes that have been considered are 18, 20, 22, 24, 26, 28, 32, 48, and 72.

To achieve recognition of an Odia alphabet, the system explained in this paper is divided into two parts; one is ***'DictionaryBuilding'*** and other is ***'FindingMatch'.*** The ***'DictionaryBuilding'*** takes a directory ***'Dictionary'*** (consisting of images of Odia alphabet), undergoes through several modules to extract features from images present in ***'Dictionary'*** and the extracted features are written in an excel file, ***'DictionaryFeatures.xlsx'***. The common feature found out from the extracted features in all font sizes for each Odia alphabet (***'DictionaryFeatures.xlsx'***) is written in an excel file, ***'CommonFeature.xlsx'***. The ***'FindingMatch'*** takes an image of Odia alphabet as input and the alphabet can be in any font size of font family ***'AkrutiOriAshok-99'***; features are extracted from the input image and the extracted feature is given as input to ***'Recognition'***. The Recognition finds a correct match for the input image.

The Lenovo ideapad 310 Laptop with 64-bit Windows 10 Operating system, 4GB RAM and Intel(R) Core(TM) i5-7200U CPU @ 2.50GHz 2.70 GHz have been used for the system. The JetBrains PyCharm Community Edition 2019.1 as Integrated Development Environment (IDE) and opencv-python 4.1.1.26 libraries has been used to implement the system.

For testing, an image of Odia alphabet is given as input to the ***'FindingMatch'*** to find a correct match. Nine font sizes have been considered for this research and 200 images of Odia alphabet of each font size making a total of 1800 images are provided as input to ***'FindingMatch'*** one at a time. The percentage of correctness has been shown in Fig. 7.

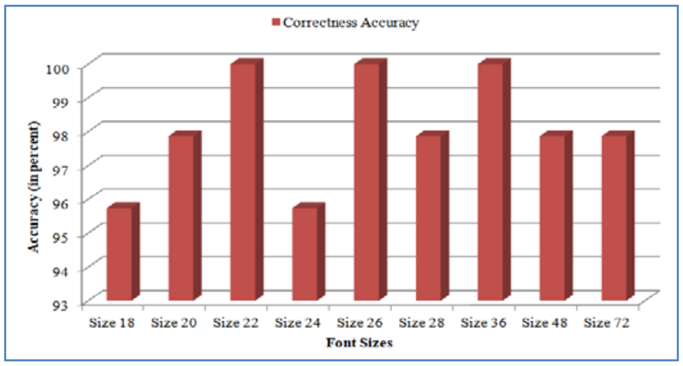

Fig. 7. Correctness Accuracy of Odia Alphabets in Different Font Sizes.

For feature extraction, [22] and [23] had used Water-Reservoir Principle to get the shapes of the characters and numerals respectively; [24] had divided the characters into nine zones and traced the shapes in each zone; [25] had found out the centroid of the character and then the angle between the centroid and the pixel to trace the shapes of the characters; and [26] had first found out some low-level strokes to detect the high-level strokes and using these strokes, the shapes of the character had been traced. The proposed approach has also traced the shapes of the characters by first dividing the character into four quadrants and then scanning each quadrant in different directions to get the features in string format. The proposed system has also been compared with the systems in [22], [23], [24], [25] and [26], and the results have been tabulated in the Table I.

TABLE I. ACCURACY COMPARISON OF PROPOSED APPROACH WITH OTHER APPROACHES

| **Accuracy Achieved by the Approaches in Related Work** | | | | **Accuracy Achieved by the Approach in this Paper** |
|---|---|---|---|---|
| **Approaches** | **Language** | | **Accuracy** | |
| [22] | Handwritten Odia | Isolated Characters | 98.6% | 98.1% |
| | | Two-Character Touching Components | 96.7% | |
| | | Three-Character Touching Components | 95.1% | |
| [23] | Handwritten English Numerals | | 94.8% | |
| [24] | Printed Odia | | 92% | |
| [25] | Printed Odia | | 91.3% | |
| [26] | Printed Gujarati | | 96.87% | |

It has also been found that alphabet Chota U ( ଉ ) is recognised as Bada U ( ଊ ) in some font sizes because they have very little difference in their structure. The system faces the same challenge for the alphabets Ra ( ର) and Ru ( ଋ).

## V. CONCLUSION

The approach described in this paper goes through two parts. First part deals with building a dictionary and the second part deals with finding a match for the image given as input. In the first part (***'DictionaryBuilding'***), a dictionary of images consisting of alphabets in the font family ***'AkrutiOriAshok-99'*** and in different font sizes are prepared. Then features are extracted from the images and written in an excel file, ***'DictionaryFeatures.xlsx'***. LCS has been used to find the common feature from the extracted features and the common feature has been written in an excel file, ***'CommonFeature.xlsx'***. The second part deals with finding a match for the image that is given as input. In second part, features are extracted from the input image and matched with the feature present in ***'CommonFeature.xlsx'***. In some cases, if more than one match or no match is found then the four quadrants of the input image have been scanned in another direction. The overall correctness accuracy of the system has been achieved as 98.1%.

As the proposed approach recognises Chota U ( ଉ) as Bada U ( ଊ) and Ra ( ର) as Ru ( ଋ) in some font sizes, hence, further research can be done in future to eliminate this disadvantage. Elimination of this problem may increase the accuracy percentage. Moreover, research can be done to reduce the number of phases of the proposed system which may increase the efficiency of the system.

## REFERENCES

[1] Devabrata Kar, Chabila Madhu Barnabodha, Published by Odisha Book Emporium.

[2] Pandit Narayan Mohapatra, Sridhar Das, Sarbasara Byakarana, ISBN: 8186085009, Published by New Students' Store.

[3] Aradhana Kar, Sateesh Kumar Pradhan, "A Three-Phase Noise Removal Approach to Achieve Accuracy in Line Segmentation of Odia text", 19th OITS International Conference on Information Technology (OCIT), pp. 54-59, 2021.

[4] "Image Thresholding, Image Processing in OpenCV" (Web Search)

[5] Ravishankar Chityala, Sridevi PudiPeddi, Image processing and Aquisition using Python", CRC Press Taylor & Francis Group, Chapman & Hall, pp 141 – 143.

[6] Nobuyuki Ostu, "A Threshold Selection Method from Gray-Level Histograms", IEEE Transactions on Systems, Man, and Cybernetics, Volume 9, Issue 1, pp 62 – 66, January 1979.

[7] S. Sridhar, "Digital Image Processing", Oxford University Press, pp 10 – 11, 2013.

[8] Rafael C. Gonzalez, Richard E. Woods, Digital Image Processing, Pearson, pp 55 – 65, Third Edition.

[9] Mark S. Nixon, Alberto S. Aguado, Feature Extraction & Image Processing for Computer Vision, Elsevier, Academic Press, pp 37 – 41, Third Edition.

[10] "os.path", https://docs.python.org/3/library/os.path.html

[11] Eric Gazoni, Charlie Clark, "openpyxl - A Python library to read/write Excel 2010 xlsx/xlsm files", Version 3.0.10.

[12] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, "Introduction to Algorithms", The MIT Press, Tata McGraw Hill Book Company, pp 350 – 355, Second Edition.

[13] Udit Agarwal, "Algorithms Design and Analysis", Dhanpat Rai & Co (P) Ltd, pp 262 – 270, Second Edition.

[14] Narasimha Karumanchi, "Data Structures and Algorithms Made Easy", CareerMonk Publications, pp 373 – 375.

[15] Steven S. Skiena, "The Algorithm Design Manual", Springer, Second Edition, pp 650 – 653.

[16] Lekh Raj Vermani, Shalini Vermani, "An Elementary Approach to Design and Analysis of Algorithms", Primers in Electronics and Computer Science, Vol. 4, World Scientific, pp 174 – 185, 2019.

[17] Jan Erik Solem, "Programming Computer Vision with Python: Tools and Algorithms for Analyzing Images", O'Reilly, pp 7 – 8.

[18] "NumPy" (Web Search)

[19] "Matplotlib.pyplot" (Web Search)

[20] Zed A. Shaw, "Learn Python 3 the Hard Way A Very Simple Introduction to the Terrifying Beautiful World of Computers and Code", Addison-Wesley, Exercise 34 (Lists), pp 120 – 121 and Exercise 39 (Dictionaries), pp 140 – 144.

[21] Kent D. Lee, "Python Programming Fundamentals", Springer, 2014.

[22] N. Tripathy, U. Pal, "HandWriting Segmentation of Unconstrained Oriya Text", Proceedings of the 9th International Workshop on Frontiers in Handwriting Recognition, IEEE Computer Society, 2004.

[23] U. Pal, A. Belaid, Ch. Choisy, "Touching numeral segmentation using water reservoir concept", Pattern Recognition Letters 24, pp 261-272, 2003.

[24] Dibyasundar Das, Ratnakar Dash and Banshidhar Majhi, "Odia Compound Character Recognition Using Stroke Analysis", Computational Intelligence in Data Mining, Advances in Intelligent Systems and Computing, volume 556, pp 325 – 332, 2017.

[25] Debananda Padhi, Debabrata Senapati, "Zone Centroid Distance and Standard Deviation Based Feature Matrix for Odia Handwritten Character Recognition", Proceedings of the International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA), Springer-Verlag Berlin Heidelberg, pp 649 – 658, 2013.

[26] Mukesh M. Goswami, Suman K. Mitra, "Printed Gujarati Character Classification Using High-Level Strokes", Proceedings of 2nd International Conference on Computer Vision & Image Processing (Volume 2), Advances in Intelligent Systems and Computing (Volume 704), Springer, pp 197 – 209, 2017.

[27] Mukesh M.Goswami, Suman K. Mitra, "Classification of Printed Gujarati Characters using Low-Level Stroke Features", ACM Transactions on Asian and Low-Resource Language Information Processing, Volume 15, Issue 4, Article 25, pp 1 – 26, June 2016.