2009

Handwritten Devanagari Character Recognition

FINAL YEAR PROJECT

SAURAV KAYAL, MEGHNAD SAHA INSTITUTE OF TECHNOLOGY

Table of Contents

Abstract

Optical Character Recognition is a procedure to recognize handwritten or printed characters from documents, automatically by machine, to transform them into digitally editable documents. The problem of Optical Character Recognition (OCR) can be considered to be subdivided into two major parts:

- segmentation of the document
- character recognition.

The main focus of this project is to work on the area of character recognition using a cognitive approach, which is both simple and less resourceful than other techniques already in use for the same purpose centering on neural networks and expert systems.

In the project, the document has been assumed to be scanned or photographed so as to generate a JPEG image file of the character to be recognized. Algorithms have been developed to address the problems of recognition of an appropriate threshold to differentiate between foreground pixels and background pixels, binarization of the image, noise removal, thinning, scaling, trimming the whitespaces, statistical and momentum-based feature extraction and storage of those extracted features.

An algorithm has been developed to identify an appropriate threshold value for each document dynamically depending on the exact values of its pixels. This threshold value helps to differentiate between the foreground and background pixels of the document. The next stage employs a simple and obvious logic for binarization of image by identifying the foreground and background pixels by comparing their values with that of the threshold found in the previous step and representing them by ‘1’ and ’0’ respectively.

In this project, for noise removal simple algorithms are used. We have restricted ourselves to one-pixel and two-pixel based noise. In the thinning phase, an algorithm is used to remove redundant pixels from the image. For this purpose, we have also used a mask which produces zero redundancy in the image. We have used algorithm to scale up/down the image. However here it is used to produce an image of the size 30x30 pixels. The size helps in logical division of the image which is needed for the feature extraction.

In the feature extraction phase, we worked on two types of features:

1. Statistical Feature: no. of horizontal zero crossing, no. of vertex points, no. of joining points, position of vertical bar etc.
2. Momentum-based Feature: average no. of black pixels, skewness, kurtosis & normalized skewness & kurtosis of the image.

Extraction of these features and storing them in a database helps to uniquely identify the character based on some recognition procedure as used in neural network.

The result obtained is satisfactory considering time and resource constraints, but needs refinement in some of the areas as mentioned in details in the following text. Verification on a larger set of samples and the corresponding statistical analysis, if can be taken up as follow up, is likely to generate more refinement and bring out more issues that can be solved then.

Introduction

This project has been taken as a part of fulfillment of the syllabus of B.Tech, Computer Science and Engineering. This section majorly deals with the background and justification of the problem, areas where the solution can be applied and the approach taken to handle the problem.

Background Review

Optical character recognition is one of the oldest problems that the computer scientists have been dealing with. Initially the goal was to recognize machine printed texts of uniform font and size. However, with initial success, the goal expanded to handling texts with varied fonts and varied size of characters. Still later, came the challenges of recognizing hand-written texts followed by texts in cursive, both machine printed and hand written. The various families of character recognition are as shown in the following figure.

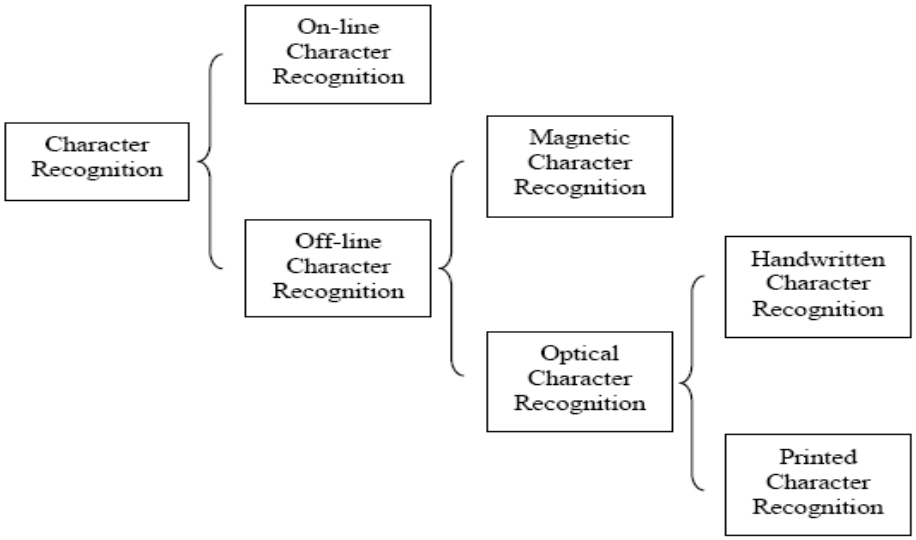


Fig 1. The different families of character recognition

Figure 1 shows the different families of character recognition. Two different families are included in the general term of character recognition

- On-line character recognition
- Off-line character recognition

On-line character recognition deals with a data stream which comes from a transducer while the user is writing. The typical hardware to collect data is a digitizing tablet which is electromagnetic or pressure sensitive. When the

user writes on the tablet, the successive movements of the pen are transformed to a series of electronic signal which is memorized and analyzed by the computer.

Off-line character recognition is performed after the writing is finished. The major difference between on-line and off-line character recognition is that on-line character recognition has time-sequence contextual information but off-line data does not. This difference generates a significant divergence in processing architectures and methods. The off-line character recognition can be further grouped into:

• Magnetic character recognition (MCR)

• Optical character recognition (OCR)

In MCR, the characters are printed with magnetic ink. The reading device can recognize the characters according to the unique magnetic field of each character. MCR is mostly used in banks for check authentication. OCR deals with the recognition of characters acquired by optical means, typically a scanner or a camera. The characters are in the form of pixelized images, and can be either printed or handwritten, of any size, shape, or orientation. The OCR can be subdivided into handwritten character recognition and printed character recognition.

Handwritten character recognition is more difficult to implement than printed character recognition due to the diversified human handwriting styles and customs. In printed character recognition, the images to be processed are in the forms of standard fonts like *Times New Roman*, *Arial*, *Courier*, etc.

Justification of the problem

An OCR engine is a system which can load an image, preprocesses the image, extracts proper image features, store the feature vectors in the image model library, and recognizes the image according to the degree of similarity between the loaded image and the image models. The general functioning of the OCR Engine is depicted by figure 2

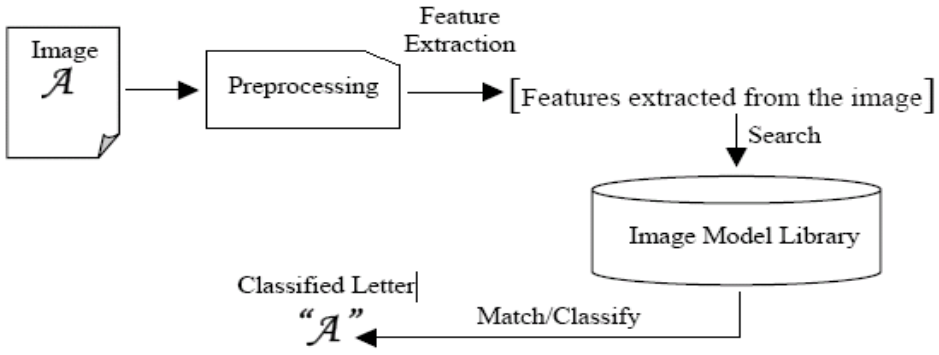


Fig 2. The basic process of an OCR Engine

This project mainly focuses on the preprocessing stage as shown in fig 2.The preprocessing stage aims to make the image be suitable for different feature extraction algorithms. The preprocessing stage which includes thresholding, binarizing, noise removal, thinning, scaling, trimming the whitespaces and so on can make the initial image more suitable for later computation.

The problem gets complicated manifold with handwritten documents where standardization of fonts and size, and recognizing a particular orientation of the words become even more complex. In the practical world, one may need to study a variety of sources, even old manuscripts, where the background to text contrast may not be high. Image defects and noise may also distort the text which makes the process of recognition more and more difficult.

Areas of application

The undertaking of this project is aimed for those who would take an attempt to the development of an OCR engine for Devanagari script. They can use it as a ready tool for the preprocessing phase of the OCR engine. Segmentation of documents is a sub-problem of the larger problem of optical character recognition. So, in order to discuss the application areas of segmentation of document, we need to discuss all the application areas of OCR.

i. **Postal addressing:** Postal addressing needs identifying handwritten addresses, which includes the segmentation and identification of recognizable characters incorporated therein.

ii. **Signature verification:** signatures are the basis of most legal, business and financial transactions and until now it has been impossible to rule out the necessity of handwritten signatures. Thus, any kind of signature verification is a possible area of application.

iii. **Handwritten bank cheque recognition:** a large number of bank cheques are still processed by hand as the identification of the details in the cheque has not yet been made possible. Successful recognition of handwriting may lead to automation in this area.

iv. **Form processing:** OCR can be used to convert handwritten forms into digital text, which saves cost and time.

v. **Note taking:** This technology bridges the gap between paper and PC by uploading data and enabling PC actions from handwritten commands on paper. It adds the capability to search within handwritten data and extract knowledge from our handwritten notes.

vi. **Forensic studies:** Handwriting plays an important role in forensic science. So ability to recognize handwriting and its key features may open new horizons in forensic science.

vii. **Maintaining old manuscripts:** Old manuscripts are a source of enormous indigenous knowledge, which can come to various uses. Old manuscripts are difficult to maintain in the libraries. Ability to recognize text by machine would enable converting these old documents to digital forms and maintain them in better condition and less space.

viii. **Diagnosis of diseases:** certain diseases like Alzheimer's disease, Parkinson's disease, can be identified beforehand from the handwriting of the victim. If developed to a considerable height, this technology can be uses for disease diagnosis purposes.

Software Requirement Specification

Overview

The project aims at developing a tool for the preprocessing of documents prior to identification of characters. The problem of segmentation of documents has been studied extensively and the problem areas identified and classified into the following heads

i. Identification of a dynamic threshold value for differentiation between foreground and background pixels.

ii. Identification of the background and foreground pixel values and their representation as '0' and '1' respectively.

iii. Noise Removal

iv. Thinning

v. Scaling

vi. Removing header lines

vii. Trimming white spaces

viii. Feature extraction & storage

ix. Recognition of the Devnagari letters.

Identification of the dynamic threshold value: The goal of this stage is to identify a threshold value dynamically that would help to distinguish between image pixels that belong to text and those that belong to the background. The threshold value identified would be completely dependent on the nature and the properties of the documents, the contrast level, the difference of contrast between the foreground and background pixels and their concentration in the document.

Binarization: This phase is intended to convert the image of the handwritten document into an array containing the pixel values in terms of only 1s and 0s, depending on whether the pixels represent text or background. This will help in the consequent modules, where only the information about the pixels of belonging to any one of the categories, text or background matters, but the exact intensity of each pixel is immaterial.

Noise Removal: Noise is the error that occurs majorly during the image acquisition process and results in pixel values that do not reflect the true intensities of the real scene. Digital images are prone to be corrupted by a variety of types of noises. An image can be infested by a variety of noise, removal of which may be a complete project in itself. We choose to remove only the most common form of noise for the convenience of segmentation of hand-written documents: 1-pixel noise & 2-pixel noise.

Thinning: Thinning is the phase where the images are standardized into a single line image, i.e. the images are thinned in such a way that they can be represented by a single line/ lines.

Scaling: Scaling is the method to scale the images is to size of 30x30 pixels. Scaling is very much important in preprocessing stage of the character recognition, because scaling helps to represent all the images in a predefined size.

Removal of header lines: This is an important phase of preprocessing stage. In this method, the header lines or 'matra' of the Devnagari letters are removed which helps to detect the positions of the vertical bar in the image (will be discussed later).

Trimming the white spaces: In this phase, the white spaces are removed so that we can obtain the minimum area required for the letters.

Feature extraction & storage: Feature extraction is the most important stage of the project. In this stage, we have found different characteristics of the Devnagari characters. These characteristics can be divided into 2 parts- a. momentum-based features and b. statistical based features. Momentum based features contain the average-pixel density, skewness, kurtosis & normalized skewness-kurtosis and statistical based features contains junction-point, open-end, horizontal zero crossing & vertical bar. We will discuss them later**.** These features are stored into database.

Recognition: In this phase, the features extracted are used to recognize the input characters using neural network.

Scope of the project

The tool developed is for facilitating developers of an OCR engine centered on Devanagari script and can be used as a readymade tool for preprocessing and feature extraction of the documents so that efforts can be focused on only character recognition in later phases of research.

The tool developed is a desktop application and for use of one user at a time. The source-code is also provided along-with to be used as a functional library for the tasks mentioned above.

Assumptions and dependencies

The suitability of the document for application of the algorithms that would be developed in the project is important. Hence a discussion is needed on the assumptions and dependencies that draws the line of limitations of the algorithm and leads to the criteria that makes the document most suitable for processing.

Quality of scanning:

The digital image of the document is prepared by scanning the original document. The quality of scan is an important parameter to measure the appropriateness of a document to be successfully recognized by OCR.

The bit-depth is the number one factor to improve OCR accuracy level. If the image pixels can be represented by grayscale (say 8-bit), or better, then OCR accuracy is found to be the best. The absolute minimum baseline for scanning is 300 dpi, below which the OCR accuracy drops considerably. The lower the quality of the scanned image, the better is the chance to improve OCR accuracy.

Condition of the Original document:

The condition of the original document from which the digital image is created for study is an important factor in defining the quality of the job. Following are the probable factors that can cause decline in the performance.

- Low contrast between the background and the text.
- If the document is torn or foxed.
- If the document is marked otherwise.
- If the document has crease
- If show-through is present.
- Cleanliness of the document.

Uniformity:

The more uniform is the text and print quality, the better will be the performance. Specially texts with non uniform characters are likely to pose more problems, which is an obvious issue for processing handwritten documents.

Nature of Printing:

The nature of the text in the original document is an important criterion for assessing the appropriateness of the document for the character recognition purpose. If the text is written in a poor manner, has broken characters, have faded characters or those with indistinct edges processing becomes difficult.

Complexity of formatting:

The text might be formatted in various complex ways in case of both printed and handwritten texts, for e.g. texts in the newspaper are presented in multiple columns. Such situations further complicate the process of document analysis.

Size of the image:

Any image taken as input here is converted here into 30x30 image as it helps in feature extraction phase.

Definition, acronyms, standards and abbreviations

Background pixel: the pixel that represents a part of the background (non-written part) of the hand-written document

Binarization: conversion of the image of a handwritten document into a format of '0' and '1' where 0 represents background pixels and 1 represents foreground pixels.

Character: a single alphabet in the hand-written document

Foreground pixel: the pixel that represents a part of the written areas of the hand-written document.

Image: the impression of the hand-written document or a part of it after segmentation taken in the JPEG format.

Noise: corrupted values of pixels which does not represent the actual information about the image.

Pixel: the smallest indivisible part of an image.

Threshold: a grayscale value that marks the line of distinction between the range of grayscale values for foreground and background pixels.

Header-line: an important feature of the Devanagari script usually called "matra".

Zone: image is divided into 3x3 logical matrixes. Each room of the matrix is known as a zone.

Vertex Point: It is the open end point(s) present in the image.

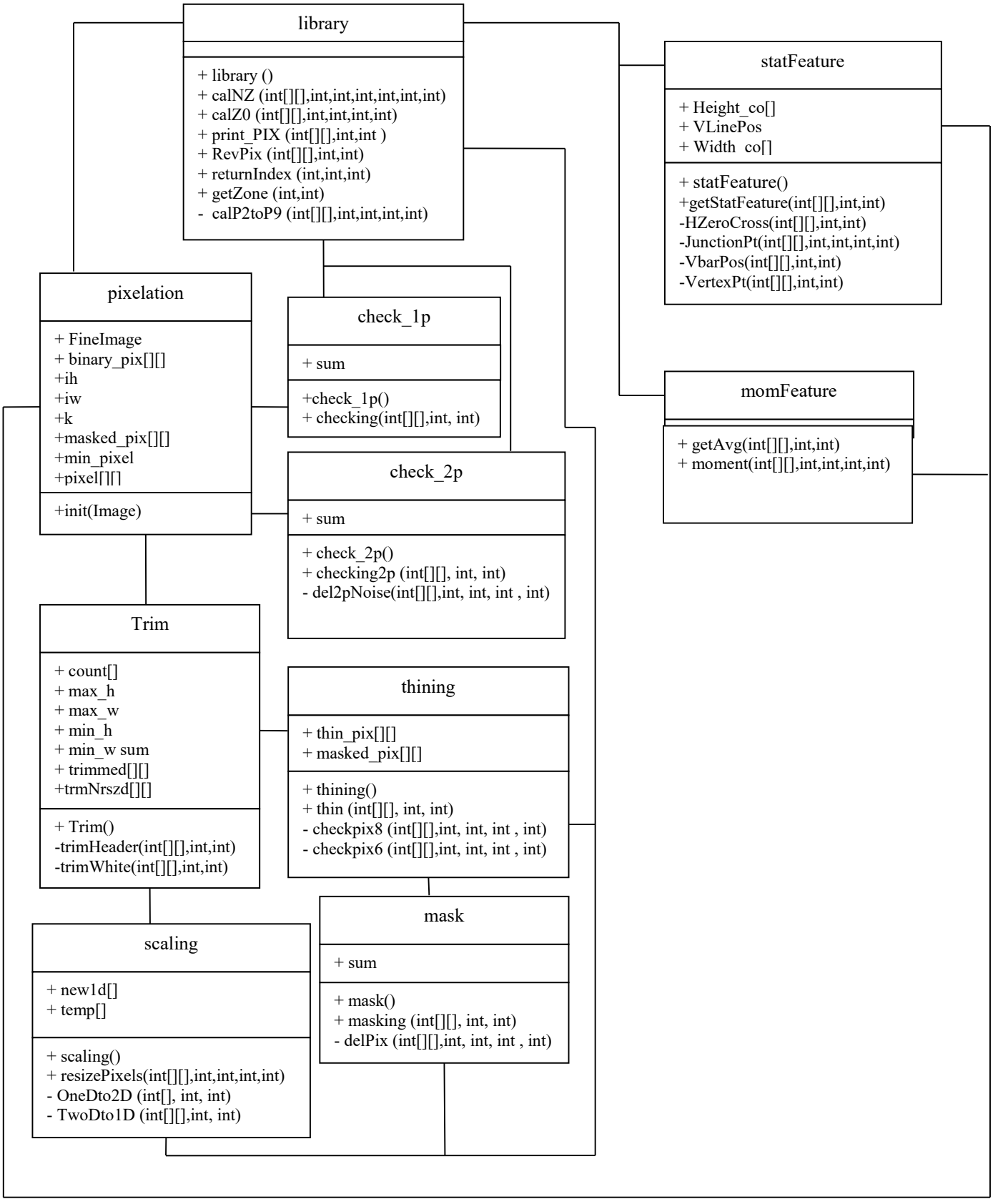
Junction Point: It is the point where two or more than two line intersects each other**.**

Vertical bar: It denotes the lines parallel to conventional y axis.

Technologies used

- The project has been developed on Java™ platform using J2SDK 1.6 standard by Sun Microsystem Inc.
- The IDE used for the purpose is NetBeans IDE 5.0 of Sun Microsystem Inc.
- The GUI has also been developed using NetBeans IDE 5.0 of Sun Microsystem Inc.

Specific Requirements - Class diagram



Use case diagram

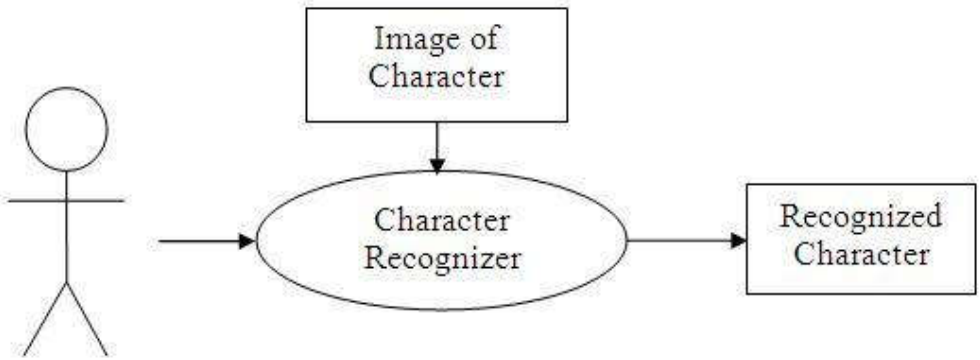


Fig 3. use case showing recognition of the input letters.

User Interface

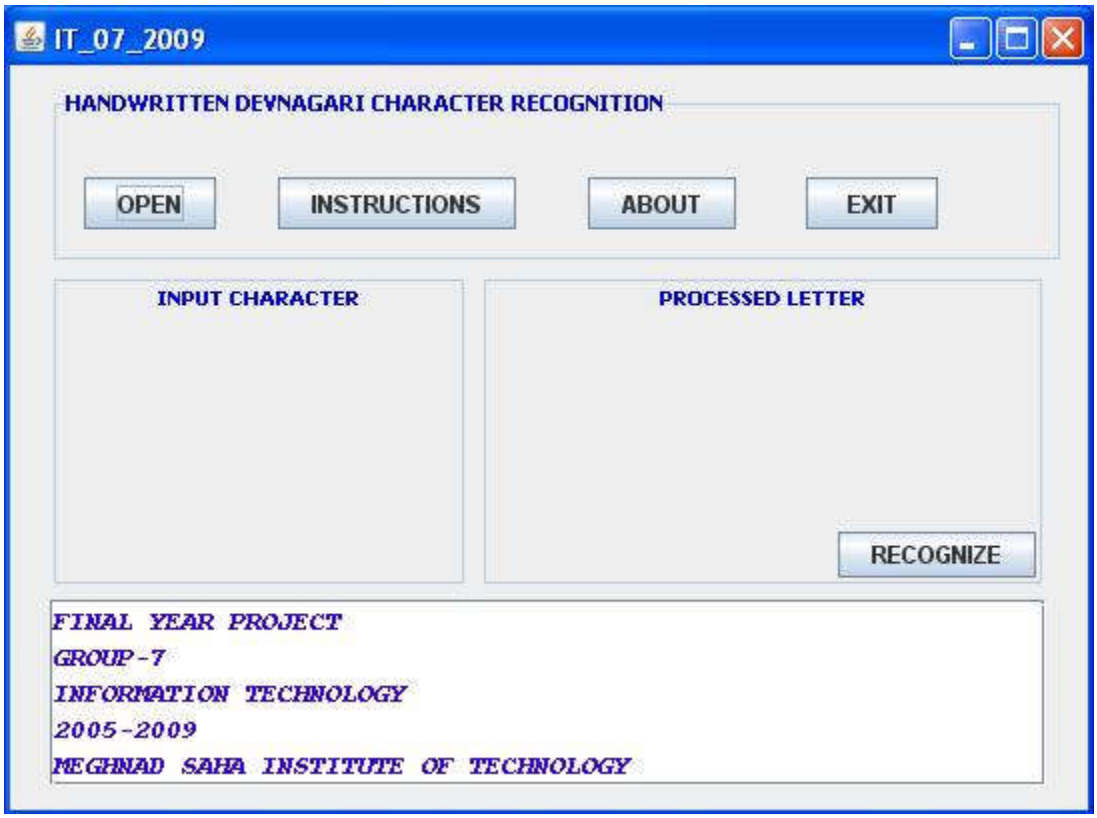


fig 4 . The opening page of the GUI

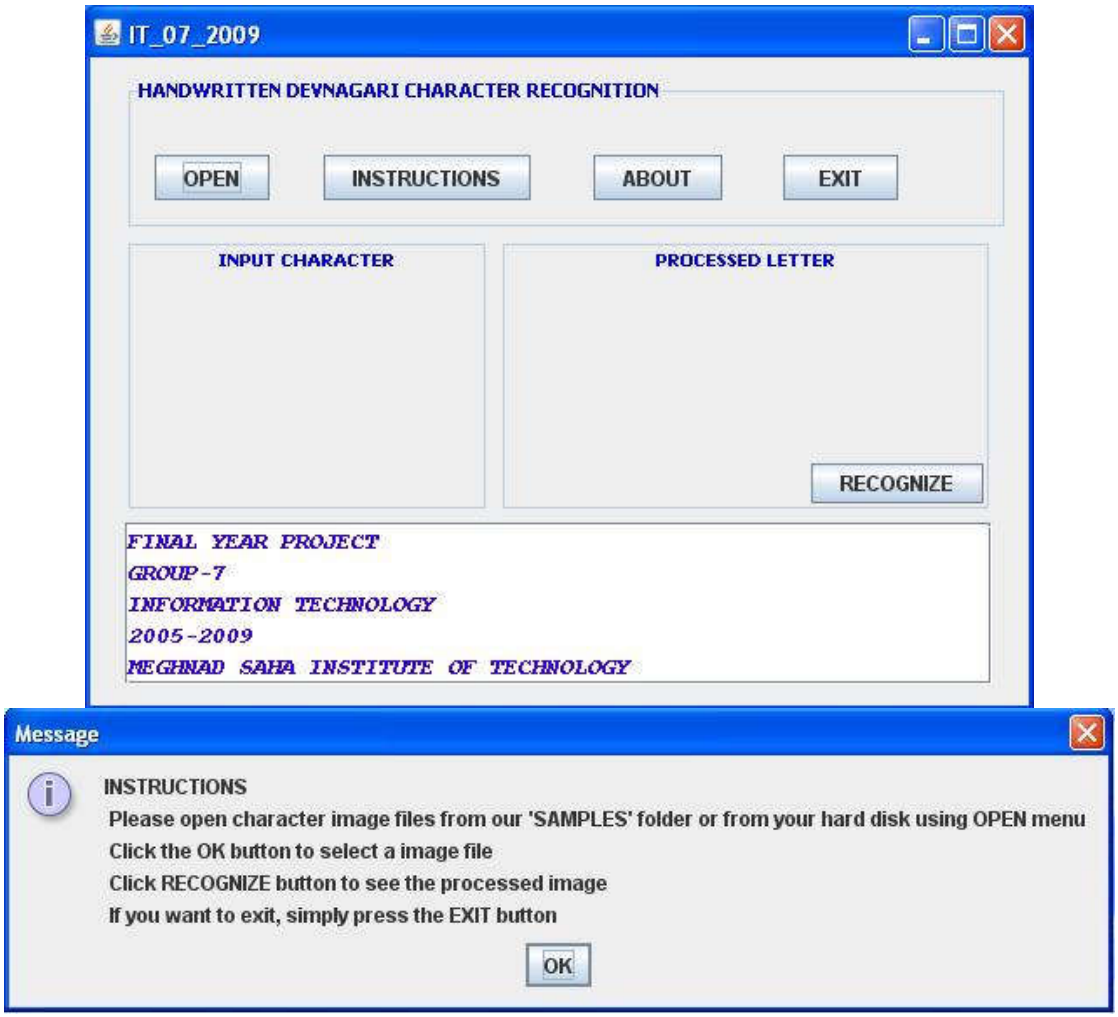


fig 5. The instruction to handle the software

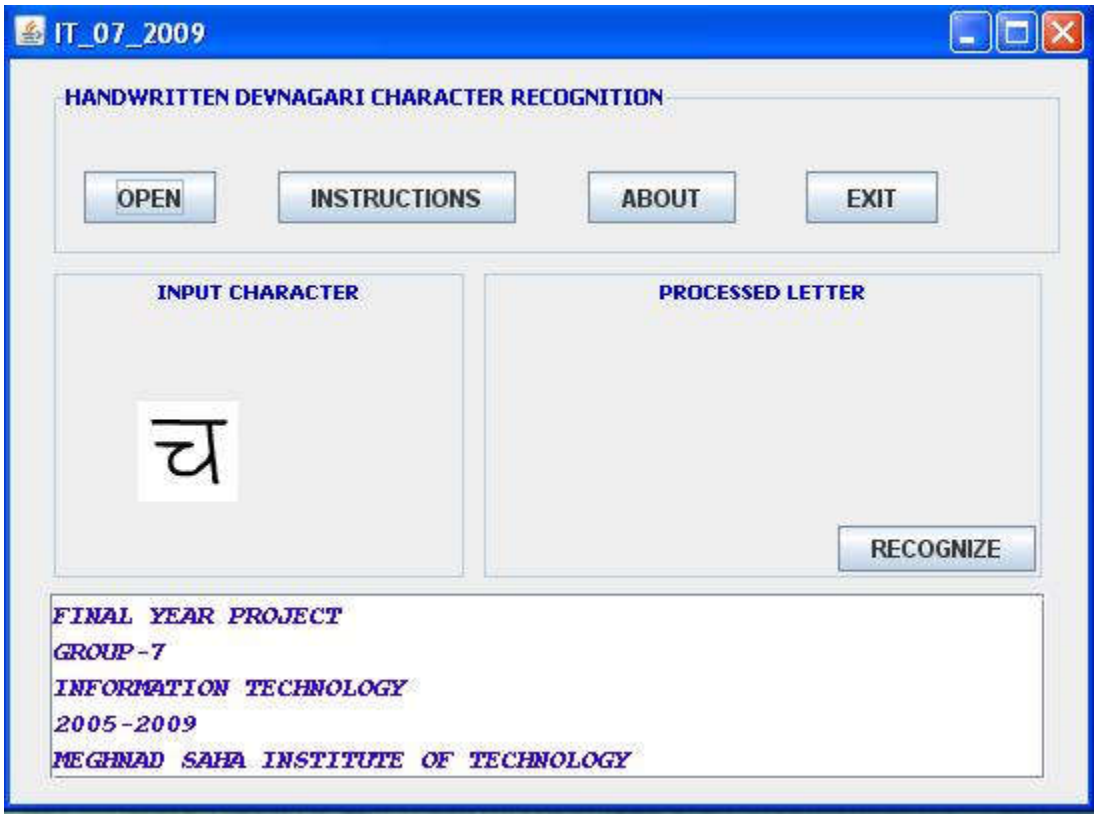


fig 6. The character is loaded to recognize

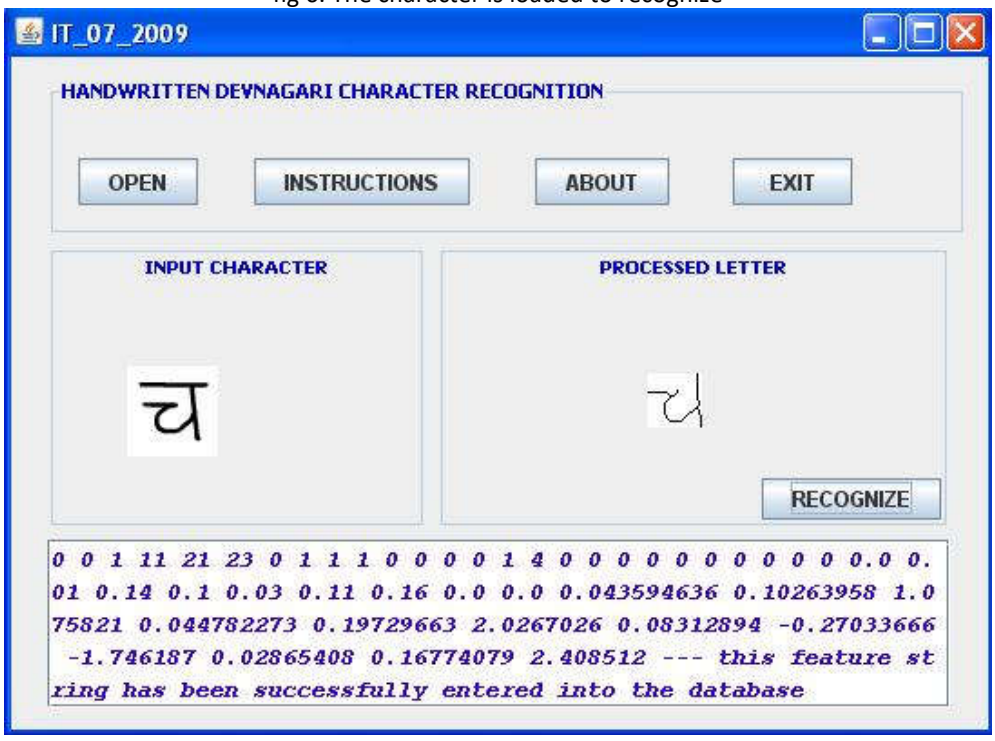


fig 7 . The character is processed & features are saved into the database

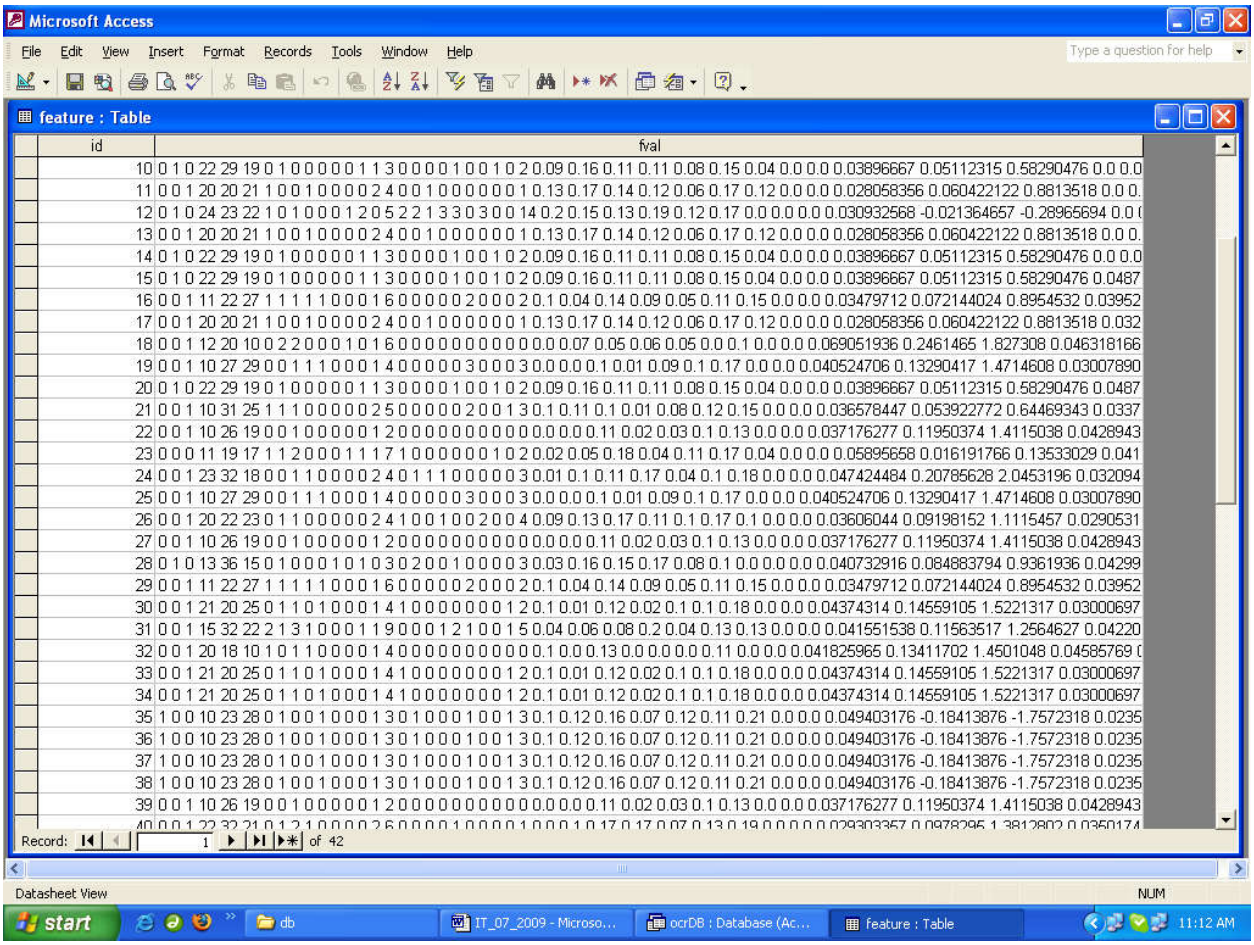


fig 8. The features stored in the database

Algorithm and Discussion

Storing the image pixels into a two-dimensional array

Algorithm – 1

Description: This algorithm is for storing the pixel-values from an image into a 2D array.

1. Get the image by scanning the handwritten document.
2. Grab the pixel values from the image and store them into a one-dimensional array. For this, 'pixelgrabber' method in Java™ is used. This method stores the pixel values directly into a one-dimensional array.
3. Using iteration store the one-dimensional array in a two dimensional array.

Thresholding

Algorithm – 2

Description: This algorithm is for identifying the threshold value of the image pixels to differentiate between foreground (text) and background pixels using an iterative mechanism. This would generate the threshold dynamically for each page depending on the color, contrast and intensity of the pixels.

1. Use iterative mechanism to find out the minimum value of the present pixels in the image.
2. Calculate the threshold value using the formula, threshold= 10% of the minimum value of pixel present in the image.

Discussion: Here 10% of the minimum pixel value is taken to optimize the output so that the character or the text can be recognized well in the presence of noise.

Binarization:

Algorithm – 3

Description: This algorithm can be used for Binarization of the image. This uses the threshold value returned by algorithm 2 to mark the background pixels as 0 and the foreground pixels as 1.

1. If pixel value < threshold, then 1 else 0.
2. This array is used by a lot of other functions in the following sections.

Discussion: This is done to categorize clearly each pixel as representing foreground or background pixels, which is the most relevant information in the context of this project and is the utmost necessity for future processing rather than concentrating on the minor details of the information carried by the values of the pixels.

Scaling

Algorithm -4

Description: As this the next phase of binarization, so it is very much obvious that we are going to use output of the previous phase as the input of this phase. This algorithm takes the binarized matrix of the previous phase and scales it accordingly.

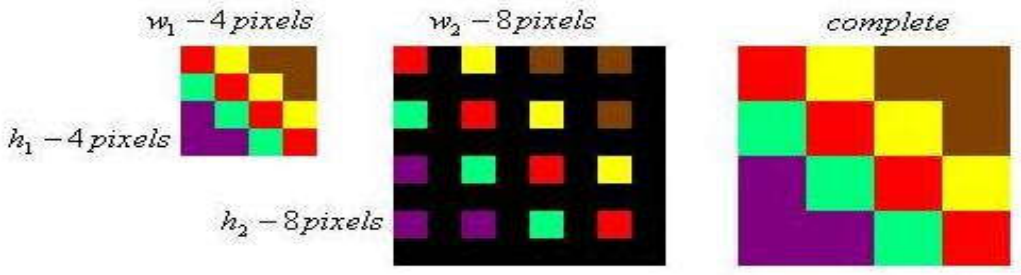


Fig .9

1. Convert the binarized 2d array into a 1d array.
2. Create an empty space to store the new scaled array (temp).
3. Calculate ‘x_ratio’ and ‘y_ratio’ using the following formulas :

   x_ratio=w1/w2

   y_ratio=h1/h2

4. For each pair of (i,j) for the resized matrix calculate the appropriate (px,py) pair in the original (pixel) matrix using the following formula:-

   px = Math.floor (j*x_ratio)

   py=Math.floor (i*y_ratio)

5. Assign the appropriate value to the new matrix using the value of (px,py) & (i,j) pair using the following formula:-

   temp [(i*w2) +j] = pixels [(int) ((py*w1) +px)]

Note:

1. This algorithm can be used for scale up as well as also for scale down.
2. We can also use this one to scale up or down any image to a certain size.
3. This method is extremely helpful in the character recognition, because it helps to convert each and every image into a standard size.

Thinning

Algorithm – 5

Description: For thinning a 'basic unit' of 3x3 pixels (fig. --) is considered in the binarized image. The thinning algorithm works on this 'basic unit'.

P3	P2	P9
P4	P1	P8
P5	P6	P7

Fig. 10– Labelling Point P1 and its Neighbours.

1**.** According to the figure above the pixel to be deleted is the P1. Its neighbours are labelled as shown above.

2**.** P1 is deleted if the following condition satisfies:

- The number of non-zero neighbours of P1[termed as NZ(P1)] lies between 2 & 6(including the border values); and
- The number of zero to non-zero transitions in the ordered set P2,P3,P4,…,P9,P2[termed as Z0(P1)] is equal to 1; and
- P2 P4 P8 =0 or Z0(P2) ≠ 1; and
- P2 P4 P6 =0 or Z0(P4) ≠ 1

3. This procedure is repeated for the whole image.

4. Now, for better result step 3 & step 4 should be repeated infinite times until no changes occur (but in this project to save computational resources we have repeated these fifty times which also assures that no further changes are occurring).

Masking

Algorithm – 6

Description: Although we have treated this as a separate method but actually it is a sub part of the previous algorithm. It assures that no single redundant pixels are present in the image in order to provide ultimate thinning of the image and also without any loss information. It represents a line as sequence of '1's with width=1.

In this algorithm also the 'basic unit' described earlier is used. The decision to delete the P1 is taken based on the following scenarios

- Don't remove centre pixel:

-	1	-
1	1	1
-	1	-

Fig .11

- Remove centre pixel:

-	1	-
1	1	-
-	-	0

-	1	-
-	1	1
0	-	-

-	-	0
1	1	-
-	1	-

0	-	-
-	1	1
-	1	-

Fig .12

Checking 1-pixel noise & 2-pixel noise

Description: Noise removal is very much important in the image processing. Noise can be of different types. Here we are dealing with 2 main type of noises- 1-pixel noise & 2-pixel noises. One pixel noise is an unintended single pixel in the image, which is surrounded by "0" pixels. This may be represented by graphically as follows-

Fig.13 -1-pixel noise

0	0	0
0	1	0
0	0	0

The 2-pixel noises are those which not surrounded by 0 pixels within a 5x5 zone which may be represented by following matrix-

Fig. 14- 2-pixel noise

0	0	0	0	0
0	0	0	0	0
0	0	1	1	0
0	0	0	0	0
0	0	0	0	0

Checking 1-pixel noise

Algorithm – 7

This algorithm removes the 1-pixel noises from the image.

1. Take the 2 D Array (thin_pix) as an input and as well as the height (h) and width (w) of that image.

2. Take every 3X3 matrix in the image iteratively and calculate the value of NZ (No. Of Non-Zero Neighbours) for the central pixel.

3. If NZ=0 Then the central pixel is considered as a 1-pixel noise and it is changed to zero in order to correct it.

Checking 2-pixel noise

Algorithm – 8

This algorithm can be used to remove the 2-pixel noise from an image.

1. Take the 2 D Array (thin_pix) as an input and as well as the height (h) and width (w) of that image.

2. LOOP START

Start checking from i=2 to i<h-3

And j=2 to j<w-3

4. If thin_pix[i][j]==1

Then go to step 4

5. If thin_pix[i][j] is connected to any other thin_pix[k][l] (where the value of thin_pix[k][l]==1) and also if thin_pix[k][l] is connected to any other 1

Then go to step 5

Else go to step 6

6. Store thin_pix[i][j]= thin_pix[i][j]

7. Store thin_pix[i][j]=0

   END LOOP

Discussion:

1. Here we have taken all the border pixels into account. Those imaginary pixels produced during the programming which can give 'array index out of bound exception' are simply ignored here to give the consistent results.

2. Here we are taking 2 D array as our input to this program; we get this 2 D array after doing the thinning & masking process. Here each time we are considering a 5 by 5 matrix to check the 2-pixel noise.

Fig. 15 An image of 'gha'

Removing header lines

Algorithm -9

Description:

This algorithm is necessary for sound processing of the image in the feature extraction phase. Consider image of 'gha'(Fig 15). Here the 'matra' (header line) is extended to left which may produce erroneous result on extraction of the feature 'Position of the vertical bar'. If the Header line is not removed then extracted feature may indicate the presence of 'middle bar' instead of 'end bar'.

1. Count the no of 1s for each row of the image and store the count values in a 1d array of size equals to the height of the matrix. For example count of 14^{th} row is stored in array [14].
2. Determines the max count.
3. Check what are values stored in the array is greater than or equal to k% of the max count (k is assumed to be 95% on experimental basis).
4. Set entire X_i row to 0 in the original 2d binarized matrix where array [X_i] satisfies the condition stated in step 3 (i = 0, 1, 2, 3...).

Trimming white spaces

Algorithm – 10

Description: we need this to find out the minimum spaces required by the letters.

1. Find out the extreme left point in the image.

2. Find out the extreme right point in the image.

3. Find out the extreme upper point in the image.

4. Find out the extreme lower point in the image.

5. Store the pixels in a new 2 d array which are lying in the range extreme left to extreme right point & extreme upper to extreme lower point.

Calculation of the statistical features:

These are features found in the image that are entirely based on position and number. Four types of features are dealt here: Position of Vertical bar, Number of horizontal zero crossings, Number and Position (in zone of the 3x3 logical divisions) of Vertex Points & Junction Points.
For the following algorithm we are going to use a 3x3 logical division (fig.16) of the images and calculations are done based on the zones.

Zone0	Zone1	Zone2
Zone3	Zone4	Zone5
Zone6	Zone7	Zone8

HZone0
HZone1
HZone2

VZone0	VZone1	VZone2

fig.16 3x3 Logical zones, Logical horizontal zones, Logical vertical zones

1. Calculating the position of vertical bar

Algorithm – 11

Description: The number of '1's is counted for each column in the binarized image matrix.

1. Maximum Count is identified and its corresponding 'column index' is noted.
2. If the value of the Maximum count is greater than or equal to k% of the image height then the vertical bar exists (k is assumed to be 40% based on experimental basis).
3. Determine the vertical zone where the 'column index' lies.
4. A 1d array is taken and one of its indexes is set to '1' based on the determination of the logical zone and rests are set to '0'. If no bar exists then all indexes are set to '0'.

2. Calculating the horizontal zero crossing

Algorithm – 12

Description: In this method the zones are going to be used are 3 horizontal zones.

1. Scan a single zone.
2. Increase a counter if the previous pixel is 0 and the current pixel 1.
3. After completion of a single zone store the value in a 1d matrix according the Horizontal zone number and set counter to 0.
4. Repeat steps 1, 2 and 3 for the three zones.

3. Calculating the positions & numbers of vertex points e.g. open-end points

Algorithm – 13

Description: This algorithm counts the total number of vertex points for each zone of the 3x3 division as well for the entire matrix. In this algorithm an array of size 10 is used to denote the number of vertex points present in each zone; first 9 indexes of the array for 9 zones and the sum of first 9 indexes are stored in the last index of the array to denote the total.

1. Scan the entire image.
2. If the current pixel is equal to 1 then check whether NZ (current pixel) =1 or not, if not then skip that pixel and check the same for the next.
3. Retrieve & store all the 9 neighbouring pixels (fig. a) including current pixel in a temporary array.
4. Now, check for 1 in the temporary array (ignore the current pixel as it is always 1) and calculate actual 'position' of that pixel in the original binarized matrix.
5. Now, check NZ ('position') =2 or not. If yes, the 'zone no' corresponding to the 'position' is retrieved and the counter of that 'zone no' is increased in the array (any one index of the first 9).
6. The sum of first 9 is stored in the last index.

Note: 'NZ' is described in the Thinning algorithm

4. Calculating the number of junction points

Algorithm – 14

Description: this algorithm is used to find out the no of joining points in a binarized image. The algorithm will search for the 1's in the image & find it whether it is a joining point. It will consider the following cases as joining point.

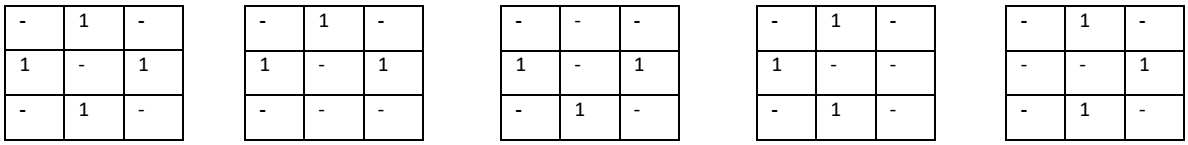


Fig.17 junction points

In each case the centre pixel is considered as the joining point.

1. Travel each pixel of the image
2. If the pixel is lying at the border, (m and n are the counters)

   If, pixel [m-1] [n]==1 & pixel[m][n-1]==1 & pixel[m+1][n]==1

   Consider pixel[m] [n] as the joining point.
3. If the pixel is not lying at the border, then,

   If , (pixel[m-1][n]==1 & pixel[m][n-1]==1 & pixel[m+1][n]==1 & pixel[m][n+1]==1)

OR (pixel[m][n-1]==1 & pixel[m-1][n]==1 & pixel[m][n+1]==1)

OR (pixel[m][n-1]==1 & pixel[m+1][n]==1 & pixel[m][n+1]==1)

OR (pixel[m][n-1]==1 & pixel[m-1][n]==1 & pixel[m+1][n]==1)

OR (pixel[m-1][n]==1 & pixel[m][n+1]==1 & pixel[m+1][n]==1)

Then the pixel can be considered as the joining point.

m-1, n-1	**m-1, n**	**m-1, n+1**
m, n-1	**m, n**	**m, n+1**
m+1, n-1	**m+1, n**	**m+1, n+1**

Fig. 18

Calculation of the momentum-based features:

1. **Calculation of the pixel density in the image**

Algorithm – 15

Description: In this phase, the average pixel density of the image is measured. This average pixel density in different parts of the image can be features of different Devnagari letters.

1. Divide the binarized image (30x30 2d array consisting of only 0's & 1's) into nine 2d arrays.
2. Calculate the no of pixels denoting 1 in each small 2d array.
3. Calculate the total no of pixels in each small array.
4. Find out the average using the formulae-

   Average pixel density in 1 small array = (total no of pixels denoting 1)/ (total no of pixels in a small array).
5. Store all the average values obtained in the above methods in an 1d array avg [].

2. **Calculation of the skewness, kurtosis & normalized skewness-kurtosis of the binarized image**

Algorithm – 16

Description: Skewness, Kurtosis and normalized skewness & kurtosis are the measures of the dispersion of a set of values w.r.t a fixed value. We have taken the centre pixel of the image (14, 14) as the fixed

point. Thus skewness, kurtosis and normalized skewness & kurtosis will be measured for those pixels which denote 1.

The formula for the skewness, kurtosis & normalized skewness kurtosis are as follows-

Skewness= (momentum of order 4)/ (momentum of order 2) ^2

Kurtosis= (momentum of order 3)/ (momentum of order 2) ^1.5

Normalized skewness kurtosis= (momentum of order 3)/ (momentum of order 4) ^0.75

This method can be subdivided into 2 parts-

a. In this method the image is divided into 2 parts- up & bottom.

b. In this method the image is divided into 2 parts – left & right.

For each of the above methods the following processes are applied-

1. Consider 3 double variables- m2, m3 and m4.
2. m2= momentum order of 2

   m3= momentum order of 3

   m4= momentum order of 4
3. For each half, the m1, m2 & m3 are calculated using the following formula-

```
m2=m2+ (j-14)*(j-14);
m3=m3+ (j-14)*(j-14)*(j-14);
m4=m4+ (j-14)*(j-14)*(j-14)*(j-14);
```

Where j= no of columns

4. Calculate

```
Skewness=SV= ((float) m3/ (float) Math.pow (m2, 1.5));
Kurtosis=KV= ((float) m4/ (float) (m2*m2));
```

Normalized skewness & kurtosis=NV= ((float) m3/ (float) Math.pow (m4, 0.75))

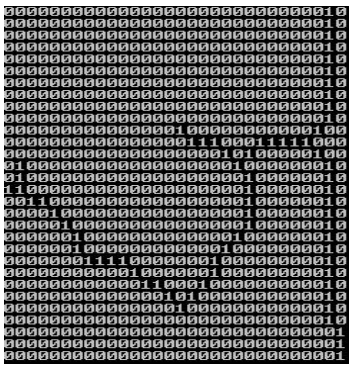
5. Store the values obtained from the above process in 4 arrays – up [], down [], left [], right [].

Result obtained

The user will give the handwritten character to the software & the program will give an output of the processed image.

Input:  fig 19

Output for the user:  Fig .20

Output produced by the software in the binarized form: fig. 21 

Discussion on result

The major target of the project was to identify a set of features of the Devnagari characters to recognize a hand-written character document. A total of 16 algorithms have been developed for the purpose, the detail discussion of which is given in articles 4.1 through 4.11. From figures 19 through 21, it is clearly seen that the image has been converted into the binarized image, i.e. it is converted in to a 2d binary matrix. Moreover, there is need to execute these algorithms on a larger set and variety of sample data to make the findings more dependable and refined.

References

[1] Giorgos Vamvakas, Optical Character Recognition for Handwritten Characters

[2] Arora, S.; Bhatcharjee, D.; Nasipuri, M.; Nagpur, L.M., A Two Stage Classification Approach for Handwritten Devnagari Characters

[3] ----Jain, Thinning Algorithm, Digital Image Processing.

[4] Thinning Mask by Mrs. Sandhya Arora, Professor, Meghnad Saha Institute of Technology.

[5] Nearest Neighbour Image Scaling. http://tech-algorithm.com/index.php?post=27

[6] Veena Bansal and R.M.K. Sinha, A Devnagari OCR and A Brief Overview of OCR Research for Indian Scripts

[7] Reena Bajaj, Llipika Dey and Santanu Chaudhury, Devnagari numeral recognition by combining decision of multiple connectionist classifiers, *S¯adhan¯a* Vol. 27, Part 1, February 2002, pp. 59–72. © Printed in India.