Optical Character Recognition For Bangla Documents Using HMM

Md. Sheemam Monjel and Mumit Khan
Dept. of CSE, BRAC University, Dhaka, Bangladesh.
sheemam@bracuniversity.net, mumit@bracuniversity.net

Abstract

In this paper we have described an OCR program made for Bangla documents. This program uses HMM for the recognition process. The description of full OCR program is too large to present here. So, we have given emphasis on the important and Bangla specific processes of the OCR pro-gram. We have defined some features of Bangla characters and described their extraction process.

1. Introduction

So far, all the prominent works related to optical character recognition for Bangla documents use artificial neural networks for the recognition process [2], [4], [9], [10]. The advantage of HMM approach over ANN approach in optical character recognition is that it can be easily extendible to the recognition of handwritten characters. Here, the features required for HMM recognition process are defined and their extraction process is described briefly.

2. Steps of OCR

There are 4 major steps in OCR system.

1. Preprocessing.
2. Feature Extraction.
3. Recognition.
4. Post processing.

The Preprocessing step prepares the document image for feature extraction. It has several sub-steps.

a) Conversion to gray-image: The original image is converted into a gray-image. Gray level of each pixel is determined by the average of red, green and blue levels of that pixel.

b) Noise reduction: The image is passed through a smoothing process to remove the noise in it. We have used a non linear smoothing algorithm for this [6].

c) Binarization: The gray-image is converted into black & white image. The foreground and background gray-levels are chosen according to the context of the image [6].

d) Skew correction: If the page is not scanned with the proper alignment, the lines will not be horizontal. So, we have to rotate the image to make the lines horizontal for further processing. Now, we are correcting the image if the skew angle is at most 15 degrees.

e) Segmentation:

i. Line segmentation: from the b&w image, horizontal blackPixel frequencies are calculated for every row. Then the lines are segmented by the rows having blackPixel frequency of 0.

ii. Word segmentation: In Bangla script, words are separated by blank spaces. So, by analyzing the vertical blackPixel frequencies of a line image, words are easily separated through the columns having blackPixel frequency of 0.

iii. Letter segmentation: Here comes the real challenge. In a Bangla word, all the letters except a few are joined by an overhead line called matra.

The letters that aren't joined by matra are listed below:

এ ঐ ও ঔ ঙ ঞ ঃ ং

The letters that are partially joined by matra are :

ঋ গ প থ ণ শ

The letters that have a matra but are not connected with it are :

ত ত্ত

Some letters always vertically overlap the adjacent letter :

ি ী ৌ ঁ ু ূ

Some letters often vertically overlap the adjacent letter :

ে ৈ

Example of vertical overlapping :

ত্রে কু কি কী

So, the matra is very useful to extract the words but it affects letter extraction process adversely. As the matra is present in some letters and absent in some

others, removing the matra altogether may cut some portion of letters. Sometimes removing the matra causes some letters to divide into pieces. So, we are preserving the matra as it is and analyze the word image avoiding the matra region for letter segmentation. The first step is to determine the matra. It is the horizontal line whose blackPixel frequency is highest among all the rows of a segmented line of script. So, the vertical location of the matra is determined during line segmentation. Middle region of a line is bounded by the matra and a horizontal line whose distance from the bottom limit of the line is equal to the distance between the matra and the upper limit of the line. The bottom 5/6 of this middle region is analyzed to separate the letters vertically. Vertical blackPixel frequencies are determined in this region and the columns having blackPixel frequencies of 0 separates the letters. In this segmentation process, the letters that vertically overlap the adjacent character outside the middle region are cut into 2 pieces. These pieces are combined together in the post processing step that comes after the recognition step.

f) Skeletization: The separated letters are thinned to have lines of only one pixel width [11].

g) Component extraction: The connected components are separated.

After the 7th step, we get the skeletized components of characters. Then features are extracted from the components. These features are used for recognition.

For using HMM, we need a sequence of objects to traverse through the state sequence of HMM. So, we have to shape the features into a sequence of objects. Hence, for each character component, we are making a tree of features and finally the prefix notation of the tree is applied to the HMM. In the tree, the number of child of a node is not fixed. So, we are using child-sibling approach to make the tree. Hence the prefix notation of the tree will contain nodes in the order: root, prefix notation of the tree rooted at its child, prefix notation of the trees rooted at the child's siblings from left to right order.

3. Features

Features are the lines of the skeleton of character components and the points joining the lines.

4. Data Structure

A tree data structure is used for storing the feature elements. Each node of the tree represents either a line or a joining point. Suppose a line L1 meets other two lines (L2, L3) at a point. Then the point will be child of L1, L2 will be sibling of the point and L3 will be sibling of L2. The directions of lines are classified into 8 main categories and they are numbered as shown in Figure 1.

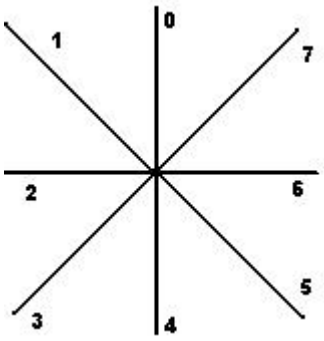

Figure 1: Directions of lines

We use an array dir[][]={{1,2,3},{0,-1,4},{7,6,5}} to hold the direction numbers. The direction of a line from a point P1 to an adjacent point P2 is retrieved readily from the array element dir[P2.x-P1.x+1][P2.y-P1.y+1]. Two adjacent lines are unified whenever the difference between their directions is 45 degrees. So, unified lines have two directions. The data structure for lines and joining points are as follows.

```
class imageComponent{}
class joiningPoint extends imageComponent
{
   int lineCount;
   line sibling;
   joiningPoint(int n){lineCount = n;}
   boolean equals(joiningPoint p)
      {return lineCount == p.lineCount;}
}
class line extends imageComponent
{
   int direction1,direction2;
   Point start;
   boolean isObsolete;
   joiningPoint child;
   line sibling;
   line(Point p){start = p; direction1=direction2=-1;}
   void setEndPoint(Point p)
   {
       isObsolete = (start.x==p.x && start.y==p.y);
```

```
        if(!isObsolete)
        {
                int max = dir1>dir2?dir1:dir2;
                int min = dir1<dir2?dir1:dir2;
                dir1 = max;
                dir2 = min;
        }
    }

boolean isAppendable(Point from, Point to)
{
        int dir[][]={{1,2,3},{0,-1,4},{7,6,5}};
        int lineDir=dir[to.x-from.x+1] [to.y-from.y+1];
        if((lineDir == dir1) || (lineDir == dir2)) return true;
        else if(dir1<0){dir1 = lineDir; return true;}
        else if(dir2<0)
          {
            if((lineDir==(dir1+1)%8)||
               (lineDir==(dir1-1+8)%8))
                {
                  dir2 = lineDir;
                  return true;
                }
                else return false;
          }
        else return false;
    }
    boolean equals(line l)
    {
        return (dir1==l.dir1 && dir2==l.dir2);
    }
}
```

5. Line Tracing and Making the Feature Tree

The steps required for making the feature tree:

1. Initialization:
   curPixel = left top corner point.
   mark curPixel as traced.
   curLine = line starting from curPixel.
   Q = an empty queue.

2. pack all the untraced black neighbors of curPixel into pointArray and mark them as traced.
   count = no. of untraced black neighbors.
   if (count=0) then goto step 3.
   else if (count=1) then goto step 4.
   else goto step 5.

3. if (Q is empty) then line tracing completed.
   else
      curLine = pop line from Q.
      curPixel = starting point of curLine.
      goto step 2.

4. if pointArray[0] is appendable to the curLine then
      curPixel = pointArray[0].
      Update the direction of curLine.
   else
      set the endpoint of curLine as curPixel.
      Create a joiningPoint and set it as the child of current line.
      Create a new line starting at pointArray[0] and enqueue it into Q.
      set this new line as the sibling of the joiningPoint.
      curLine = pop line from Q.
      curPixel = starting point of curLine.
      goto step 2.

5. set the endpoint of curLine as curPixel.
   Create a joiningPoint and set it as the child of current line.
   Create new lines starting at the points in pointArray and enqueue them into Q.
   set the first new line as the sibling of current line and make a sibling chain with all the new lines.
   curLine = pop line from Q.
   curPixel = starting point of curLine.
   goto step 2.

In this algorithm, the black neighbors of the endpoint of current line are enumerated. If only one black neighbor is found, the point is checked to insert into the current line. If the current line is unidirectional and the direction to new point is only 45 degree away from direction of current line then the new point is appended to the current line and the direction of current line is updated to hold the new direction. If the current line is compound and it has two directions then if the direction to new point conforms to one of the directions then it is appended to the current line. If the neighbor cannot be appended, then a new line is started from this point.

Finally, when the endpoint of a line is set, the line is marked as obsolete if its endpoint is same as the starting point. These obsolete lines are deleted from the feature tree.

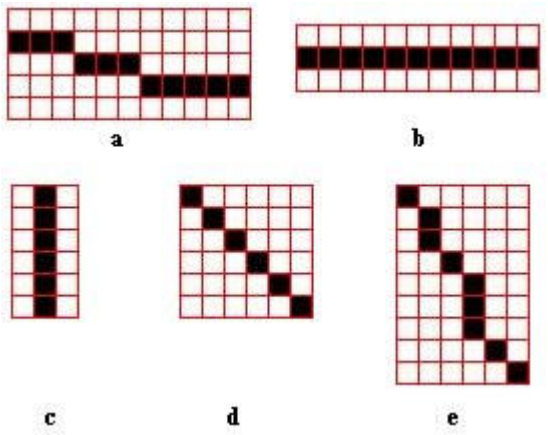

Figure 2: Lines of different directions

Line 2(a): direction1 = East, direction2 = South-East
Line 2(b): direction1 = East, direction2 = Nil
Line 2(c): direction1 = South, direction2 = Nil
Line 2(d): direction1 = South-East, direction2 = Nil
Line 2(e): direction1 = South-East, direction2 = South

6. An example

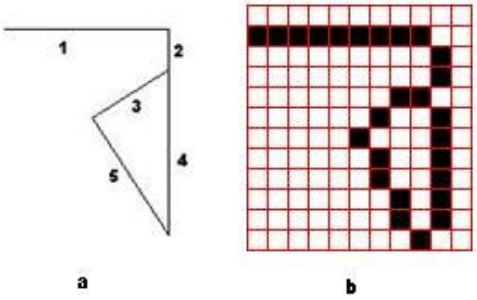

Figure 3: A Bangla letter
(a) the letter skeleton (b) corresponding bitmap

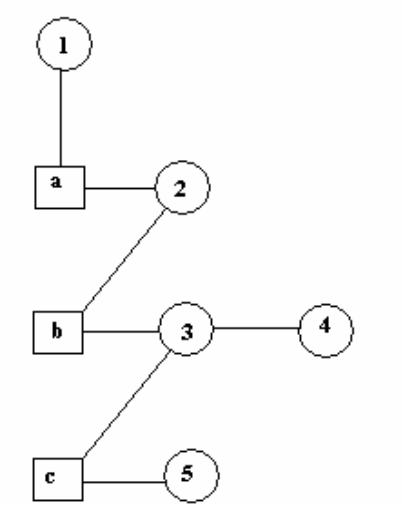

Figure 4: The feature tree corresponding to the letter of Figure 3. A circle denotes line and a rectangle denotes joining point.

If we represent line by L(dir1,dir2), then
line1 = L(6,5)
line2 = L(4,3)
line3 = L(3,2)
line4 = L(4,-1)
line5 = L(5,4)

If we represent joining Point by P(siblingCount), then
point 'a' = P(1)
point 'b' = P(2)
point 'c' = P(1)

Now if we express the tree in prefix notation, we will get
L(6,5) P(1) L(4,3) P(2) L(3,2) P(1) L(5,6) L(4,-1)
This representation is used in HMM for recognition.

7. Recognition

Once the sequence of objects for the letter components are available, it is very easy to train the HMM and use it for recognition.

8. HMM

Hidden Markov Models (HMMs) are used for both online and offline character recognition systems for different scripts around the world [7],[8]. A Markov chain or process is a sequence of events, usually called states, the probability of each of which is dependent only on the event immediately preceding

it. An HMM represents stochastic sequences as Markov chains where the states are not directly observed, but are associated with a probability density function (pdf). The generation of a random sequence is then the result of a random walk in the chain (i.e. the browsing of a random sequence of states Q = {$q_{1, \ldots,}$ q_k}) and of a draw (called an emission) at each visit of a state. The sequence of states, which is the quantity of interest in most of the pattern recognition problems, can be observed only through the stochastic processes defined into each state (i.e. one must know the parameters of the pdfs of each state before being able to associate a sequence of states Q = {q1, …, q_k} to a sequence of observations X = {x1, …, x_k}). The true sequence of states is therefore hidden by a first layer of stochastic processes. A Hidden Markov Model is defined by specifying 5 things :

i. Q = the set of states = {q_1, q_2, …, q_n}

ii. V = the output alphabet = {v_1, v_2, …, v_m}

iii. $\pi(i)$ = probability of being in state q_i at time t = 0 (i.e., in initial states)

iv. A = transition probabilities = {$a_{i,j}$}, where $a_{i,j}$ = *Pr*[entering state q_j at time t + 1 | in state q_i at time t].

v. B = output probabilites = {$b_j(k)$}, where $b_j(k)$ = P*r* [producing v_k at time t | in state q_j at time t].

The recognition process measures the probabilities of generating the object sequence got from the feature extraction of the component under consideration from the HMMs of different letter components . The highest among these probabilities decides the letter component with the closest match. The process of calculating these probabilities is described below.

Let the object sequence corresponding to a letter component be X = {x_1, x_2, …, x_n}. and the HMM corresponding to a template letter component be H. Now, probability of generation of X from H is

$p(X|H) = \Sigma\, p(X,Q \mid H)$, for every possible Q.
[Q is state sequence {q_1, q_2, …, q_T}].

Now, according to Bayes theorem,

$p(X,Q \mid H) = p(X,Q \mid H) \,.\, p(Q \mid H)$

The terms of right hand side are calculated by:

$$p(X,Q \mid H) = \Pi\, p(x_i \mid q_i,H) \text{ , } i = 1 \text{ to } T$$
$$= b_1(x_1).b_2(x_2)\ldots b_T(x_T).$$
$$p(Q \mid H) = \Pi\, a_{i,i+1} \text{ , } i = 1 \text{ to } T\text{-}1$$
$$= a_{1,2}.a_{2,3}\ldots a_{T-1,T}$$

While comparing the lines in HMM, only the directions are checked for matching. And in the case of comparing the joiningPoints, the no. of lines originating from it is considered.

9. Postprocessing

Now, after the recognition process, some post processing is necessary. It has two steps.

9.1. Constructing the letters from components.

In the segmentation process, some letters are cut into pieces because the overlap their adjacent letters. So, after recognition, the pieces should be joined together to form the original letter. Consider the following example.

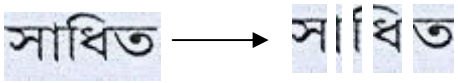

Figure 5 : Extraction of letters from a word

In this example, a word is separated into 5 parts. But in the 4th part, there are two components. So, after the recognition process, we will get 6 objects :

1. consonant letter ssa (স)
2. vowel sign A-kar (া)
3. vowel sign A-kar (া)
4. upper portion of vowel sign I-kar (ি).
5. consonant letter dha (ধ)
6. consonant letter ta (ত)

Now, we have to unify the object3 and object4 to make vowel sign I-kar (ি).

9.2. Rearranging the letters.

Sometimes, the letters of a word are not stored (in files) in the order they appear on the screen after rendering. In Unicode standard, dependent vowels or vowel signs come after the consonant associated with it. But on the screen as well as in the printed documents, the vowel signs may appear on left, down or right side of the consonant Vowel signs may also

wrap the consonant up from both sides. So, after a word is recognized, the letters are rearranged according to Unicode standard[12]. For example, when we will get e-kar (ে) and a-kar (া) around a character, we should make it o-kar (ো). The same rule should be applied to make ou-kar (ৌ).

10. Conclusion and Future Work

There are still some problems regarding the letter segmentation. Sometimes adjacent letters are joined to each other in such a way that they cannot be vertically separated in the normal vertical histogram approach. Some papers are avail-able describing various ways to handle the problem of segmentation of joined characters [1], [3], [5]. Our future work in this regard will be analyzing the features of joined letters and incorporating a better way of segmentation.

11. Acknowledgement

This work has been partially supported through PAN Localization Project (www.PANL10n.net) grant from the International Development Research Center, Ottawa, Canada, ad-ministered through Center for Research in Urdu Language Processing, National University of Computer and Emerging Sciences, Pakistan.

12. References

[1] A. Belaid, C. Choisy, U. Pal, "Water Reservoir Based Approach for Touching Numeral Segmentation", *Proc. 6th ICDAR*, 2001.

[2] B. B. Chaudhury, U. Pal, "OCR in Bangla : An Indo-Bangladeshi Language", *Proc. IEEE*, 2001, pp. 269-273.

[3] B.B. Chaudhury, U. Garain, "Segmentation of Touching Characters in Printed Devnagari and Bangla Scripts Using Fuzzy Multifactorial Analysis", *IEEE Transactions on Systems, Man and Cybernetics-Part C: Applications and Reviews, Vol. 32, No.4*, Nov, 2002.

[4] S. Chowdhury, S. Dutta, G. Parthasarathy, "On Recognition of Bengali Numerals with Back Propagation Learning", *Proc. IEEE*, 1992, pp. 94-99.

[5] Y. K. Chen, J. F. Wang, "Segmentation of handwritten connected numeral string using background and fore-ground analysis", *Proc. 15th ICPR*, 2000, pp. 598-601.

[6] E. Gose, R. JohansonBaugh, S. Jost, *Pattern Recognition and Image Analysis*, Prentice Hall of India, New Delhi 2002.

[7] R. Nopsuwanchai, D. Povey, "Discriminative Training for HMM-Based Offline Handwritten Character Recognition", *Proc. 7th ICDAR*, 2003.

[8] H. Nishimura, T. Timikawa, "Off-line Character Recognition using On-line Character Writing Information", *Proc. 7th ICDAR*, 2003.

[9] M. M. Rahman, A. N. M. E. Rafiq, S. Rahman, "Online Handwritten Bangla Numeral Recognition by Grid Method", *Proc. 2nd ICECE*, 2002.

[10] S. H. Shaikot, F. Kawsar, M. S. Saikat, "Bengali Digit Recognition System using 3-layer Backpropagation Neural Network", *6th ICCIT*, 2003, pp.327-331.

[11] C. Y. Suen, T. Y. Zhang, "A Fast Parallel Algorithm for Thinning Digital Patterns", *Communications of the ACM, Vol.27*, March 1984.

[12] www.unicode.org