Alphabet Recognition Using Pixel Probability Distribution

VaidehiMurarka, Sneha Mehta, DishantUpadhyay, SonaliBhadra, AbhijeetLal

ABSTRACT

The project uses techniques of Image Processing and Machine Learning in Computer Vision.

Alphabetrecognition is the mechanical or electronic translation of scanned images of handwritten, typewritten or printed text into machine-encoded text. One of the popular mobile applications includes reading a visiting card and directly storing it to the contacts.

The implementation of our project has been done using Visual Studio and Open CV (**Open Source Computer Vision**). Our algorithm is based on Neural Networks (Machine learning). The language used is c++. Our software is a fully functional model. **We have running software available with us.**

The project was implemented in three modules viz.-

1. Training: This module aims **"Database Generation**". Database was generated using two methods:

- *Run-time generation*
- *Contour–detection*:

2. Preprocessing: InputImage is pre-processed using image processing concepts such as adaptive thresholding, resizing and cropping and is made ready for segmentation. "**Segmentation"** includes extraction of lines, words, and letters from the processed text image.

3. Testing and prediction: The extracted letters are classified and predicted using the neural networks algorithm. The algorithm recognizes an alphabet based on certain mathematical parameters calculated using the database and weight matrix of the segmented image.

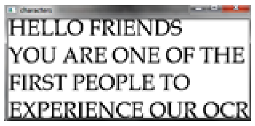
INDEX

Contour-detection, neural networks, Pre-processing, Recognition coefficient, Runtime-template generation, Segmentation, Weight matrix

---

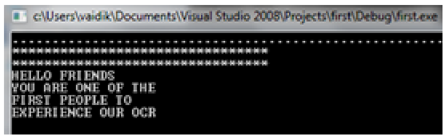
OBJECTIVE

INPUT IMAGE("JPEG FORMAT")



HELLO FRIENDS
YOU ARE ONE OF THE
FIRST PEOPLE TO
EXPERIENCE OUR OCR

OUTPUT("TXT FORMAT")



```
c:\Users\vaidik\Documents\Visual Studio 2008\Projects\first\Debug\first.exe
-----------------------------------------------------
*********************************
*********************************
HELLO FRIENDS
YOU ARE ONE OF THE
FIRST PEOPLE TO
EXPERIENCE OUR OCR
```

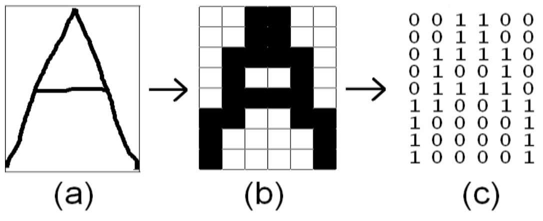
INTRODUCTION

The recognition of optical characters is known to be one of the earliest applications of Artificial Neural Networks, which partially emulate human thinking in the domain of artificial intelligence. The recognition of characters from scanned images of documents has been a problem that has received much attention in the fields of image processing, pattern recognition and artificial intelligence. Classical methods in pattern recognition do not as such suffice for the recognition of visual characters. For this reason we use the method of neural networks.

CUSTOM ALGORITHM

IMAGE DIGITIZATION

When a document is put to visual recognition, it is expected to be consisting of printed (or handwritten) characters pertaining to one or more scriptsor fonts. This document however, may contain information besides optical characters alone. For example, it may contain pictures and colors that do not provide any useful information in the instant sense of character recognition. In addition, characters which need to be singlyanalysed may exist as wordclustersor may be located at various points in the document. Such an image is usually processed for noisereduction and separation of individual characters from the document. It is convenient for comprehension to assume that the submitted image is freed from noise and that individual characters have already been located (using for example, a suitable segmentation algorithm). This situation is synonymous to the one in which a single



```
0 0 1 1 0 0
0 0 1 1 0 0
0 1 1 1 1 0
0 1 0 0 1 0
0 1 1 1 1 0
1 1 0 0 1 1
1 0 0 0 0 1
1 0 0 0 0 1
1 0 0 0 0 1
```

Fig.1

Noise-free character has been submitted to the system for recognition.The process of digitization is important for the neural network used in the system. In this process, the input image is sampled into a binary window which forms the input to the recognition system. In the above figure, the alphabet *A* has been digitized into *6X8=48* digital cells, each having a single colour, either black or white. It becomes important for us to encode this information in a form meaningful to a computer. For this, we assign a value *+1* to each black pixel and *0* to each white pixel and create the binary image matrix *I* which is shown in the Fig. (1.c). So much of conversion is enough for neural networking which is described next. Digitization of an image into a binary matrix of specified dimensions makes the input image invariant of its actual dimensions. Hence an image of whatever size gets transformed into a binary matrix of fixed pre-determined dimensions. This establishes uniformity in the dimensions of the input and stored patterns as they move through the recognition system.

LEARNING MECHANISM

In the employed system, a highly simplified architecture of artificial neural networks is used. In the used method, various characters are taughtto the network in a supervised manner. A character is presented to the system and is assigned a particular *label*. Several variant patterns of the same character are taught to the network under the same label. Hence the network learns various possible variations of a single pattern and becomes adaptive in nature. During the *training process*, the input to the neural network is the input matrix *M* defined as follows:

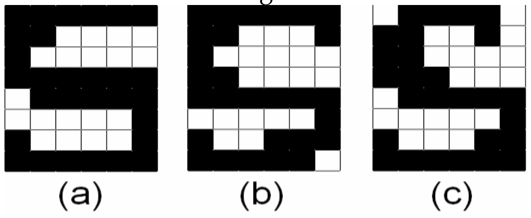
If I (*i, j*) =1*Then M*(*i, j*) =1
Else:
If I (*i, j*) =0 *Then M*(*i, j*) =(-1)

The input matrix *M* is now fed as input to the neural network. It is typical for any neural network to learn in a supervised or unsupervised manner by adjusting its *weights*. In the current method of learning, each candidate character taught to the network possesses a corresponding weight matrix. For the *kth*character to be taught to the network, the weight matrix is denoted by *Wk*. As learning of the character progresses, it is this weight matrix that is updated. At the commencement of teaching (supervised training), this matrix is initialized to zero. Whenever a character is to be taught to the network, an input pattern representing that character is submitted to the network. The network is then instructedto identify this pattern as, say, the *kth*character in a knowledgebaseof characters. That means that the pattern is assigned a label *k*. In accordance with this, the weight matrix *Wk*is updated in the following manner:

```
for all i=1 to x
{
for all j=1 to y
{
 $W k (i, j) = W k (i, j) + M (i, j)$
}
}
```

Here *x* and *y* are the dimensions of the matrix *Wk*(and *M*). The following figure shows the digitization of three input patterns representing *S* that are presented to the system for it tolearn.

Fig.2:



Note that the patterns slightly differ from each other, just as handwriting differs from person to person (or time to time) and like printed characters differ from machine to machine. All characters shall each have acorresponding weight matrix.

A close observation of the matrix would bring the following points to notice:

- The matrix-elements with higher (positive) values are the ones which stand for the most commonly occurring image-pixels.

- The elements with lesser or negative values stand for pixels which appear less frequently in the images.

Neural networks learn through such updating of their weights. Each time, the weights are adjusted in such a manner as to give an output closer to the desired output than before. The weights may represent the importanceor priorityof a parameter, which in the instant case is the occurrence of a particular pixel in a character pattern. It can be seen that the weights of the most frequent pixels are higher and usually positive and those of the uncommon ones are lower and often negative. The matrix therefore assigns importance to pixels on the basis of their frequency of occurrence in the pattern. In other words, highly probable pixels are assigned higher priority while the less-frequent ones are penalized. However, all labelled patterns are treated without bias, so as to include impartial adaptationin the system.

NETWORK ARCHITECTURE

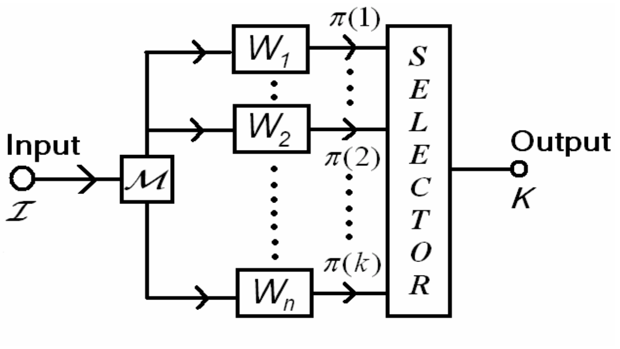


Fig.3

The overall architecture of the recognition system is shown in Fig. (3).In this system, the candidate pattern *I* is the input. The block *'M'* provides the input matrix *M* to the weight blocks *Wk* for each *k*. There are totally *n* weight blocks for the totally *n* characters to be taught (or already taught) to the system. The recognition of patterns is now done on the basis of certain statistics that shall be defined next.

Candidate Score (Ψ(k)): This statistic is a product of corresponding elements of the weight matrix *Wk*of the *kth*learnt pattern and an input pattern *I* as its *candidate*. It is formulated as follows:

$$\Psi(k)=\sum_{i=1}^{x}\sum_{i=1}^{y} Wk(i,j)*I(i,j)$$

It should be noted that unlike in the training process where *M* was the processed input matrix, in the recognition process, the binary image matrix *I* is directly fed to the system for recognition.

Ideal Weight-Model Score (◎):

This statistic simply gives the sum total of all the positive elements of the weight matrix of a learnt pattern. It may be formulated as follows (with ◎(*k*) initialized to *0* each time).

```
for i=1 to x
{
for j=1 to y
 {
if $W k( i, j) > 0$
then{$\mu (k) =\mu(k) +W k ( i, j)$
}
}
}
```

Recognition Quotient (Q): This statistic gives a measure of how well the recognition system identifies an input pattern as a matchingcandidate for one of its many learnt patterns. It is simply given by:

$Q (k) = \psi(k)/ \mu(k)$

The greater the value of *Q*, the more confidence does the system bestow on the input pattern as being similar to a pattern already known to it.

The classification of input patterns nowfollows the following trivial procedure:-

1. For an input candidate pattern *I*, calculate the recognition quotient (*Q*(*k*)) for each learnt pattern *k*.
2. Determine the value of *k* for which *Q*(*k*) has the maximum value.
3. The pattern k for which *Q*(*k*) has the maximum value is our recognised pattern.
4. Too low maximum value of *Q*(*k*) (say less than 0.5) indicates poor recognition.

To improve recognition of this particular pattern, the same pattern can be repeatedly input to the system and taught to it as before under the same label. As a result, the value of*Q* approaches unity after each time the pattern is taught. This illustrates learning from prior experience in neural networks.

For the execution of the custom algorithm, we divided the software into three modules, namely- Training, Segmentation and Testing & Prediction.

TRAINING

1) RUN TIME GENERATION OF DATABASE

A major challenge in front of us was to create various templates and use them to create our weight matrix i.e. to train our application. The storage and loading of many templates would not only occupy a lot of memory, but would also be very inefficient. So the algorithm of Run Time Template Generation was very useful.

As per this algorithm, letters from A to Z were written only once each in a text file. Now each letter was taken, and using the function **cvPutText( )**, the letter was put in a blank image. The image size was the predetermined size of the weight matrix i.e. 32 X 32. To use this opencv function, first a function called **cvInitFont( )** is used, which initializes the font type, thickness, skew and the height and width of the font. cvPutText( ) takes the pointers to the image and the character, address of font, position of letter in image and its colour as its parameters. Thus a letter is put into an image, in the desired font, size, colour and thickness.

For this algorithm, we have used 6 standard fonts inbuilt in opencv, 2 different sizes and 3 different thicknesses. Thus at run time itself, we get the templates of a single letter in 36 different ways. These templates will now be used for generation of weight matrix. We create a weight matrix of size 32 X 32 and initialize

Once we get a template of a letter at run time, we pre-process the image by using the functions **cvThreshold** and **cvSmooth**. The algorithm requires a cropped image of size 32 X 32 as input at all times. So we wrote a piece of code for cropping the template. This code finds the leftmost, rightmost, topmost and bottommost pixels of the letter and sets a **ROI** (Region of Interest) accordingly. We then resize the cropped image to the size 32 X 32, using the **cvResize( )** function. We then pass the resulting image to a function, which adds its data to the weight matrix.

In the function, a blank image of size 32 X 32 was loaded and an ROI was set somewhere near the center of the image, with height 32 and width same as the width of the cropped image. We then copied the cropped image into this ROI and then we reset the ROI. Now the modified template was sent to the weight matrix generation function.

First we had created a weight matrix of size 32 X 32 and initialized all its elements to 0. Once an image comes into this function, we create a matrix corresponding to it. We then access the pixel values of the image using **cvGet2D( )**. If the pixel is the background pixel, we set its value to -1 in the corresponding matrix. If it is a pixel of the character, we set its value to 1. We then add this matrix to the previously formed weight matrix, to form a new weight matrix. Thus for each of the 36 templates of a letter, the weight matrix goes on getting incremented. Vectors of labels and weight matrices were created, which were filled as more letters came. Now we have to store the contents of this weight matrix in a text file, which will form our database.
The format in the database is the name of the letter followed by the contents of its weight matrix. The file also contains the total number of weight matrices and the size of the matrices.
We also have a provision to add two or more such weight matrices, to make a larger database.
ADVANTAGES OF THIS METHOD:

- Saves a lot of memory and time
- Templates need not be made manually

DISADVANTAGES OF THIS METHOD

- Only a limited number of fonts are available. So we don’t get a variety in the database

2) CREATION OF A DATABASE USING BLOB DETECTION

Another method of creating the database is to carry out the letters from an image using the contours methods. According to this algorithm:

- An image containing different fonts of a same letter is an input.
- Program will extract out different all the letters from an image using contour detection method.
- The program will also draw a bounding box around the letter.
- Then we can pass the coordinates of this box and crop out an image and resize it to 32x32 .
- The program will carry out all the letters from the image input.
- These cropped images are then stored in a vector and used to create the weight matrix.

Now, how this weight matrix is generated?
This is same as discussed in the earlier topic “runtime generation of templates.

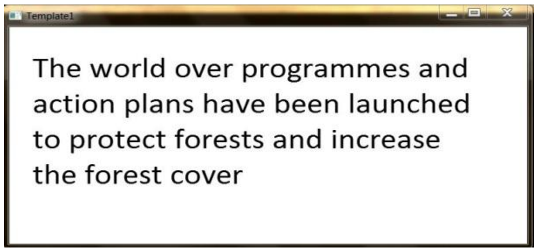
SEGMENTATION

A test image is first broken down or segmented into its component letters before the latter can be sent for classification and prediction. How we go about all this is explained below.

1. CROPPING

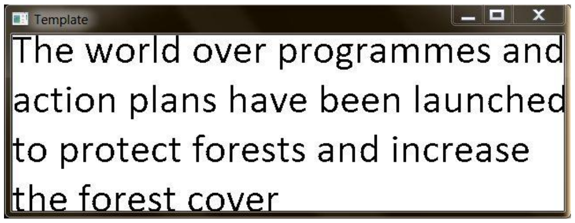
Before an image is sent for segmentation it is necessary to first crop it. For this purpose we have written a function for cropping. In this function we have assumed that background is white while foreground is black. The function can be modified a bit to suit the needs of images which have black background and white foreground. The output of this function is an image with the text touching all its four sides.

Input to cropping function


Template1

The world over programmes and
action plans have been launched
to protect forests and increase
the forest cover

Output from cropping function


Template

The world over programmes and
action plans have been launched
to protect forests and increase
the forest cover

The aim of the cropping algorithm is to set an appropriate Region of Interest (ROI) around the text. We find this ROI by finding the leftmost, the rightmost, the bottommost and the topmost black pixels in the image.

2. SEGMENTATION OF LINES

The input image is passed to the 'extractlines' function. And this function does just what its name suggests, it extracts different lines of text from the image.

But before that the image undergoes adaptive and binary thresholds to remove the existing noise from the image. After that it is resized according to its size. That is if the image height is less than 45 or the image width is less than 600 it is resized maintaining its aspect ratio as we have done in the code below.

Now the image is sent for cropping.

As we have mentioned earlier in the segmentation algorithm, first we calculate the horizontal projection of the cropped image. This means that the number of black pixels (black because here it is assumed white background and black foreground) in each of the rows of the cropped image are stored in a *vector*. Now we have a vector whose size is the same as the height of the cropped input image.

Now we iterate through the vector and locate the position of the first row which has zero black pixels, because zero black pixels means no text falls in that row which indicates a new line. Now the image is segmented (cut) from that position.

We repeat the same procedure until we cover all the rows.

The extracted lines are simultaneously passed to the 'extractletters' function. The output of the 'extractlines' function is the images of the different lines of text in the input image.

3. SEGMENTATION OF LETTERS

We will now discuss how to extract individual letters from a line.

ALOGORITHM

a) The most important step that needs to be performed before segmentation of characters is "Pre-Processing". This step includes resizing of the image so that the minimum size of the input image is greater than 32x32. Next, thresholding needs to be performed (adaptive). The average standard deviation of an image is calculated; according to it the thresholding is done. The image is also smoothened before segmentation.

b) After pre-processing, calculate the number of black pixel in each column. Traverse the image vertically, if a black pixel is encountered increment the count(of black pixels) for that column by 1. Store the values in a vector. In the code "vector ch" contains the number of black pixels in each column.

c) When we encounter a column containing some black pixels and another column besides it which is void of black pixels this means that the character has ended there. "ch.at(i)>0&&ch.at(i+1)==0" takes care of the above condition .

d) The Integer "letter" stores the number of the times condition (discussed above in 3.) is satisfied. Thus , indirectly stores the total number of characters in each line.

e) Declare two vectors namely 'r' and 's' of type "IplImage *". 's' stores the segmented character and 'r' contains the image remaining after a character has been cropped from the image.

f) Set the ROI(Region Of Interest) using the following logic. We know that when condition 3 is satisfied the character ends at 'i'th position. So , we need to crop the character from (i+1)th position. Thus, the x and y position of the first character encountered is (0,0). The height of the character will be same as the height of the line and the width is (i+1). Set the ROI and push the copied image in the vector 's'. Now, set ROI in the same image with height ="line's height" ,width="line's width-(i+1)", y=0 and x=(i+1). Push the image obtained into the vector 'r'. This condition will help us to copy the remaining portion of the image after a character has been segmented and can be used for segmenting another character. Using the image pushed in 'r' repeat step 6 till the value of the "letter-1" >=0.

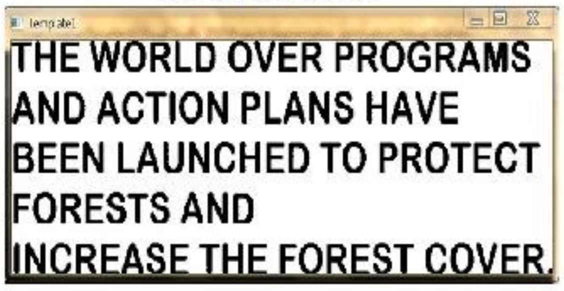
g) Integer "flag" is used for the special condition that is when the line contains a single character. We increment the value of the flag by 1 when the condition 3 is satisfied. Else the value of the flag would remain 0,as it was initialized so. If the value of the flag is zero this means that the line contains a single character and the loop discussed in step 6 will not run. Thus there will be no images in the vectors 's' and 'r'. Thus to avoid this we push the line itself into the vector 's'.

h) Finally, all our segmented characters are now present in the vector 's'. We have created a structure named "inventory" with global scope.

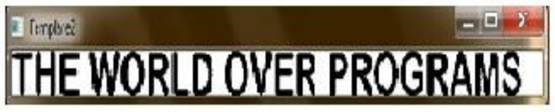
```
struct inventory
{
        int x;
        int y;
        IplImage* image;
        int label;
};
```

Declare a vector "inv" of type "struct inventory" globally. Push the segmented images ,its x and y coordinate into the vector "inv" as per the loop.
*Remember that the vector "inv" contains uncropped images of the character. Therefore.before using them further crop them using the "cropping" function. But the line you obtain from "extractlines" function are cropped ones. This will help you avoid unnecessary blank images in our vector 'inv'.

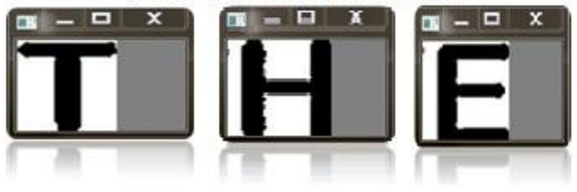
CROPPED IMAGE


THE WORLD OVER PROGRAMS
AND ACTION PLANS HAVE
BEEN LAUNCHED TO PROTECT
FORESTS AND
INCREASE THE FOREST COVER.

SEGMENTED LINE


THE WORLD OVER PROGRAMS

SEGMENTED CHARACTERS



4. SPACE DETECTION

Till now we just saw how to segment individual characters from an image. Before we move to the recognition of individual character, another important aspect is "space detection". As we all know words are separated by spaces and detecting them is important for displaying lines.
So, let's have a glance at the algorithm for "space detection".

ALGORITHM
First, calculate the average width of the line. Store the value as an integer "avg1". Pass this value calculated to the "find_spaces" function we are going to use to determine the position of spaces in the line.

Next, in the "find_spaces" function calculate the number of black pixels in each row, as we did in "extractletters" function.

Now, we need to find spaces, so, search for the condition that a column contains 0 black pixels. If the above condition is encountered, then increment the value of the integer "zcount". Store the column number as 'x1'. Keep on incrementing the value of column till we get a column which contains black pixels. Store this column number as 'x2'. The width of the space is (x2-x1).

Compare the width of the space with the character. If the width of the space is greater than or equal to 40% of (average width of the characters of the line, or 'avg1') then the space encountered is the space between words or else it is the space between the characters.

Create a structure spaces globally and a vector of type "struct spaces" globally, containing integers 'x' and 'y'.

The value of "zcount+1" will give us the actual position of te space in the line.

Push the values "zcount+1" and "line_number+1" as the 'x' and 'y' values respectively of the structure into the vector "spaces".

Finally, during the time of display using the position number of the spaces displays them along with the characters simultaneously. We shall discuss the displaying of the characters and spaces later in detail, after detection of the characters.

- **<u>TESTING AND PREDICTION</u>**
  The third module in our letter recognition software is the prediction of the segmented characters which are being generated by the segmentation module. This part is of utmost important as this shall be the final output of our application and shall be the deciding factor of the efficiency and effectiveness of our program. Hence it is of prime importance to focus on this part and implement our custom algorithm in such a way so as to obtain high predictability and satisfying results. However, it still is very difficult to predict each and every character accurately for all of its occurrences. But we definitely achieve a satisfying and deterministic output.

METHODOLOGY
As we know, we are using the weight matrix based algorithm and shall be using some predefined and pre decided parameters in order to identify a particular character. We have the weight matrices generated for each and every alphabet (capital English) which have been created using more than 500 sample training templates for each letter from A-Z. The weight matrix is a 32x32 matrix calculated for each letter. Now we use this weight matrix to calculate statistical parameter s namely:

'psi': Candidate Score
'mu': Ideal Weight Model Score
'Q' : Recognition quotient
The procedure of calculating these statistical parameters is described as follows:

1) <u>Computation of 'psi'</u>: We receive the segmented character as input for our prediction module. This black and white image is resized and converted to a matrix of 1's and 0's, 1 denoting a black pixel and 0 denoting a white pixel. We denote this input matrix by I (i, j). Psi is a statistic which is a product of corresponding elements of the weight matrix W_k of the kth learnt pattern (kth alphabet) and an input pattern I as its candidate. It is formulated as follows:

$$\boldsymbol{psi(k)} = \sum_{i=1}^{x}\sum_{j=1}^{y} \boldsymbol{W_k\,(i,j) * I(i,j)}$$

2) <u>Computation of 'mu'</u>: This statistic simply gives the sum total of all the positive elements of the weight matrix of a learnt pattern. It may be formulated as follows (with mu (k) initialized to 0 each time). It is formulated as follows:

```
for i=1 to x
{
for j=1 to y
{
if $W_k( i, j) > 0$ then
{
$mu (k) = mu( k) + W_k( i, j)$
}
}
}
```

Basically 'mu' turns out to be the sum of all the positive elements in the weight matrix for the kth letter.

3) <u>Recognition Quotient (Q)</u>: This statistic gives a measure of how well the recognition system identifies an input pattern as a matching candidate for one of its many learnt patterns. It is simply given by:
Q (k) = psi (k)/mu (k)
The greater the value of Q, the more confidence does the application bestow on input patter as being similar to a pattern already known to it.

ALGORITHM

Get the input segmented character image, preprocess it and convert it to 32x32 matrixes of 1's and 0's.

Compute 'mu' value of the kth weight matrix for each such weight matrix, in our case there would be 26 such matrices for each alphabet. Hence we get 26 'mu' values.

Compute the psi value with each weight matrix and the input matrix using the above method. Just as for 'mu', we shall get 26 such values corresponding to each alphabet.

Calculate Q value as the ratio of 'psi' and 'mu'. Q can take positive as well as negative values depending upon how close the input matrix is to the weight matrix.

The label associated with the weight matrix for which the maximum Q value occurs is identified as the matched letter and returned in the output.

This process is repeated for all the segmented characters obtained from the segmentation module.

*The weight matrices for each alphabet are stored in a text file (generated after the training phase) and can be accessed by calling the required functions.

CONCLUSION

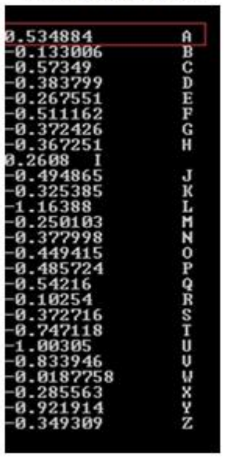
The main advantage of our software is the database generation method being used. The total size of our database is restricted only to a few kilobytes and can be easily extended for the recognition of handwritten characters. The testing and prediction phase is based on simple pixel probability. Segmentation being the most important part of our software was also executed using a simple algorithm. It provides us with a highly simplified algorithm to help recognize English characters. The software possesses immense potential and has an extensive scope. The application is efficient and deterministic to a great extent. The application is also efficient considering the time and memory constraints. As far as possible, we have used dynamic memory allocation techniques thereby improving the memory efficiency of the program. The same algorithm can be extended for handwriting recognition, by training it with handwritten samples. However, the number of training samples required would be significantly higher.

RESULTS AND DISCUSSIONS

As we can see, the maximum Q value is for the letter 'A'. Hence, the input letter would be recognized as 'A'.

INPUT IMAGE

OUTPUT IMAGE



```
0.534884          A
-0.133006         B
-0.57349          C
-0.383799         D
-0.267551         E
-0.511162         F
-0.372426         G
-0.367251         H
0.2608   I
-0.494865         J
-0.325385         K
-1.16388          L
-0.250103         M
-0.377998         N
-0.449415         O
-0.485724         P
-0.54216          Q
-0.10254          R
-0.372716         S
-0.747118         T
-1.00305          U
-0.833946         V
-0.0187758        W
-0.285563         X
-0.921914         Y
-0.349309         Z
```

ACKNOWLEDGMENT

We would like to express my gratitude to the Practice School Division of BITS Pilani for the PS1 Program.We are especially grateful to Mr. AbhayRaghuwanshi and Mr. NinadPandey from Global Logic, Nagpur for their help and advice in the project work. We are deeply indebted to Mr. AshwinKorde for his support and encouragement. We would like to thank our PS instructor Mrs. VaishnaviUnde for her constant guidance, and our Practice School co-instructor Mr. MohitAgrawal for his efforts to make our work enjoyable.

REFERENCES

www.aishack.in,
http://www.waset.org/journals/waset/v38/v38-32.pdf,
http://www.cs.berkeley.edu/~fateman/kathey/char_recognition.html,
http://opencv.willowgarage.com/documentation/cpp/index.html,
http://www.ee.iitb.ac.in/~icvgip/PAPERS/161.pdf