Building a Personal Handwriting Recognizer on an Android Device

D. Dutta
Dept. of IT
Heritage Institute of Technology
Kolkata, India
deepai.dutta@gmail.com

A. Roy Chowdhury
Dept. of IT
Heritage Institute of Technology
Kolkata, India
arunirc@gmail.com

U. Bhattacharya
CVPR Unit
Indian Statistical Institute
Kolkata, India
ujjwal@isical.ac.in

S. K. Parui
CVPR Unit
Indian Statistical Institute
Kolkata, India
swapan@isical.ac.in

***Abstract* — The wide usage of touch-screen based mobile devices has led to a large volume of the users preferring touch-based interaction with the machine, as opposed to traditional input via keyboards/mice. To exploit this, we focus on the Android platform to design a personalized handwriting recognition system that is acceptably fast, light-weight, possessing a user-friendly interface with minimally-intrusive correction and auto-personalization mechanisms. Since cursive writing on smaller screens is not usual, here we study non-cursive handwriting only. The recognition is done at character level using nearest-neighbor matching to a small, automatically user-adaptive and dynamically updating library of character-class template gestures.**

Keywords* — *Handwriting recognition; gesture recognition; template matching; nearest neighbor; Android handwriting recognition; adaptive handwriting recognition

I. INTRODUCTION

Recently tabs have become popular and these are now available with sufficient configurations suitable for common computing jobs. An increasing number of professionals now prefer to travel with the tablet instead of their laptops. This is mainly because the tabs are much more portable than the laptops or notebooks and also such a handheld device has superior battery life. These latest computers are usually equipped with various software applications that enable its user to work on documents, prepare presentations, compile spreadsheets and so on. These software applications are a way forward for professionals who need to compute on the move.

Among various tabs available in the market Android-powered tabs are relatively more popular due to its affordable price. Also, a large section of the population finds Android tabs really impressive. It is needless to say, although the Apple iPad are excellent product, they are often not affordable to the professionals of smaller organizations. On the other hand, the Android operating system was put together by Google and it has already established as a great alternative to the Apple iOS. Also, there are hundreds of applications (apps) for Android platform providing stable and powerful performance with frequent regular updates ensuring that it will survive for a long period.

Since these devices with small form factor do not have room for a physical keyboard, a virtual on-screen keyboard is provided for input method. Although a group of users have become accustomed in typing using a virtual keyboard but a larger group does not get comfort in typing using this on-screen keyboard unlike the physical one. The situation is particularly serious when it needs typing in one of the Indian scripts containing a large number of characters where pressing multiple keys for inputting a single character is frequently required. An obvious solution to this problem, particularly in the Indian scenario, is to consider handwriting as the input method. A user should find handwriting in an Indian script a more natural way of interaction than using an onscreen keyboard on the touchscreen device.

In view of the above, we developed a light-weight personalized handwriting recognition system suitable for touch screen based Android devices. There has been a lot of work on handwriting recognition in various scripts. They often use machine learning techniques such as Multilayer Perceptron (MLP) [1], Support Vector Machine (SVM) [2], hidden Markov model (HMM) [3,4] etc. along with a variant of distinguishing features. Although such recognition schemes provide acceptably high recognition performance, usually they are not light-weight enough to be suitable for hand-held devices. On the other hand, template matching schemes [5] too have application potential, especially as they can adapt dynamically to user writing-style and they have comparatively low computational complexity. Similar strategy is suitable for most mobile platforms with limited processing power.

Android 1.6 and higher SDK platforms include a new application pre-installed on the emulator, called Gestures Builder. One can use this application to create a set of pre-defined gestures for his/her own application. Our present study is based on leveraging the idea of this Gestures Builders.

In the literature [6], the following desired aspects of a "gesture shortcut system" with handwriting recognition had been discussed:

1. It should be able to predict the target for an input sample based on the targets of similar samples from the same user.

978-0-7695-4774-9/12 $26.00 © 2012 IEEE
DOI 10.1109/ICFHR.2012.189

2. It should be able to take advantage of samples drawn by other users.
3. It should be able to recognize handwriting even when there are no matching templates.

The proposed personalized handwriting recognition system is aimed at fulfilling the above criteria and it has been simulated on Android-based touchscreen platform for a set of 62 Bangla characters, consisting of the basic characters and character modifiers. A block diagram of the proposed system is shown in Fig. 1. Since here we do not consider any characteristics specific to Bangla, the same scheme should work on other scripts. Starting with a small pre-built library of online handwritten character samples, gradually the tool adapts to the handwriting style of the user and the error rate drops significantly over subsequent uses.

The remainder of our paper is organized as follows. In Section II we give a brief survey of related works available in the literature. We discussed a few characteristics of Bangla script in Sec. III. In Sec. IV-A, we describe the method of building the gesture library. The recognition details on the touchscreen are described in Sec. IV-B. Our approach for auto-learning from corrections is described in Section IV-C. The proposed application "remembers" from mistakes by inserting the wrongly recognized handwritten gesture (character) into its true class through user selection. Experimental results are provided in Section V and Section VI concludes the article.

II. Related Work

In the literature, there is extensive work on online handwriting recognition as well as gesture recognition, the latter being mostly done in the field of Human-Computer Interactions. Li [6] used gestures on Android platform to create a tool that allowed users to rapidly access data by drawing gestures on the screen. Further, Ouyang and Li [7] coupled handwriting recognition and gestures on the Android platform to better allow users to interact with their mobile phone apps via shortcuts in the form of gestures. An SVM was used to recognize the stroke classes after breaking the input handwritten words into strokes, following which character recognitions took place. Matching input gestures against learned gestures was done by an appearance-matching algorithm. Online recognition of handwritten strokes of both characters and drawings was investigated in [8, 9] for interactive table-top surfaces, enabling them to have real-time recognition of gestures and handwriting.

Benchmark recognition results of online handwritten Indian script character recognition can be found in [10]. In [11], Bhattacharya et al. studied a segmentation based analytic scheme for recognition of unconstrained online Bangla handwriting. An HMM based on context dependent sub-word units was used to recognize writer independent Bangla words in [12]. Limited lexicon unconstrained online handwritten Bangla word recognition based on a hybrid recognition technique using MLP and SVM was studied in [13].

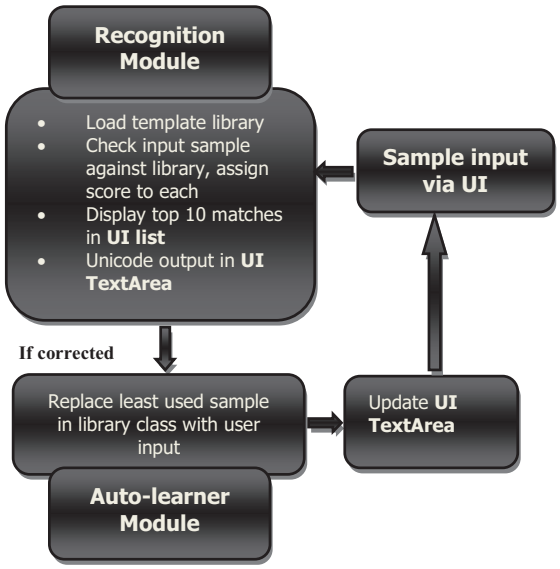

Figure 1. Block diagram of the proposed handwriting recognition system.

III. Some Characteristics of Bangla Script

Like most other Indian scripts, Bangla runs from left to right in writing and it is a mixture of syllabic and alphabetic scripts and possesses no equivalent to capital letters as in Latin scripts. It has 50 basic characters (shown in Fig. 2(*a*)) consisting of 11 vowels and 39 consonants. In addition to these basic characters, often a modified character gets attached to a basic character. Here, we include the case of 10 vowel modifiers and 2 consonant modifiers getting attached to a basic character. Shapes of these character modifiers are shown in Fig. 2(*b*).

Bangla script additionally consists of a large number of compound characters formed by merging two or more characters from the above lists. However, in the present study, we did not consider such compound characters.

In the present study, we use handwritten samples of Bangla basic characters from an existing publicly available database [10]. This database consists of a large number of samples. However, instead of adding

such a large sample database in the template library, only 15 representative samples from each class were manually chosen to build the library. More samples can be added latter to each class by a user, or as a result of the auto-learning from corrections.

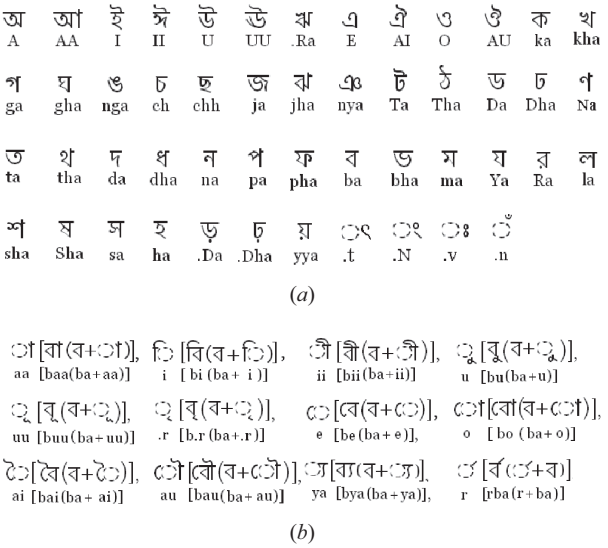

Figure 2. Ideal shapes of Bangla (*a*) basic characters, (*b*) character modifiers.

IV. The Handwriting Recognition System

Our recognition system consists of the following components:

1. A "Template Library" that stores handwriting samples as Gesture objects.
2. A "Gesture Builder" that allows us to build a library from manually input gestures, or use an existing database of samples.
3. The "Recognition" module that takes in a series of user-made gestures and outputs the corresponding Unicode characters in a text area.
4. An "Auto-learner" module (part of the Recognizer) that learns from mistakes in recognition corrected by user intervention.

A. Building the Template Library

Though the system will adapt to the user's style over usage, initially a small library is provided so that the user can immediately start using the device instead of having to build a library from scratch using his/her handwritten samples before being able to write anything meaningful. A set of 15 representative samples is considered for each character class to build the library. A few samples used in the present study are shown in Fig. 3. An interface for adding new samples to the library is shown in Fig. 4.

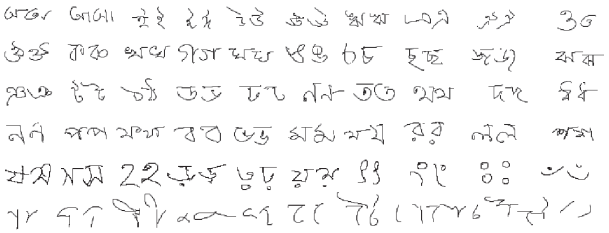

Figure 3. A few samples of online handwritten Bangla characters.

For easy and fast integration into the existing user interface of Android, we use the platform's framework in some parts instead of defining our own data structures.

First we describe a "gesture": a hand-drawn shape on a touch screen, which is represented in the Android library as a Gesture object. It can have one or multiple strokes known as GestureStroke. GestureStroke in turn consists of a sequence of timed points known as GesturePoints. Multiple GesturePoints together form a GestureStroke. These gestures can be stored in a GestureLibrary. GestureStore is an interface, which stores and maintains all the Gestures, which are present in the GestureLibrary.

Figure 4. The interface for adding new UNIPEN samples to the library.

As the popular UNIPEN format is often used to store online handwriting data samples, we describe below how to initially populate the gesture library with existing samples available in UNIPEN format. A valid UNIPEN file essentially consists of all the coordinates of the shape drawn in any device (Touch Screens, Tablets). Sample points in a UNIPEN file begins with a ".PEN_DOWN" tag and the last sample point is followed by a ".PEN_UP" tag. There may be one or more pairs of ".PEN_DOWN", ".PEN_UP" tags between the first and last sample points of a file.

Before conversion of the UNIPEN file to a Gesture object, all the UNIPEN Files are preprocessed with a

Smooth function where a 3-point moving average is applied on all the strokes having at least three sample points. Touch Screen devices come with different screen sizes and hence there is a need to scale-down those samples which are larger than the size of the target display area and also scale-up those samples which are very small compared to the display area.

Initially, we read the coordinates of all sample points of an input UNIPEN file and compute the bounding box of the character shape. Following this, a scale factor is calculated based on the dimensions of the bounding box and the display area of the device. We divide both x and y coordinates of each sample point by this scale factor and the resulting coordinates are translated so that the C.G. of the sample is shifted to the center of the display area. Now, a GesturePoint object (a standard object of JAVA used by the Android OS) is constructed for each sample point consisting of its x coordinate, y coordinate and a timestamp t in milliseconds. The GesturePoint object corresponding to the first point of a character sample uses $t = 0$ and the timestamp values of successive GesturePoints of the character are obtained by incrementing the value of t of the previous object by 10. All such GesturePoint objects corresponding to a stroke (i.e., sequence of points lying between two successive ".PEN_DOWN" and ".PEN_UP") is stored into an array and it is passed as an input to a GestureStroke class. Each such GestureStroke thus formed corresponding to a character sample is added to a Gesture object one by one. Thus, for each character sample we obtain one Gesture object consisting of one or multiple GestureStrokes. Each Gesture object has a unique identifier string, part of which identifies the particular Gesture with its true character class. These Gestures are stored within a Gesture Libraryfile, which is used during the recognition phase. A few samples of 5 different classes of our Gesture Library is shown in Fig. 5.

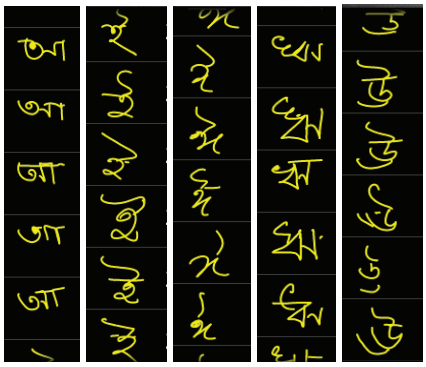

Figure 5. Five character classes and their samples from the gesture library, reflecting the wide variation in writing styles among users.

B. *The Recognition Module*

A brief description of the GUI front-end is required for full understanding of the underlying processes. It consists of the following:

1. A Gesture Overlay View where the gesture is to be drawn.
2. A "Canvas" area below the transparent overlay view
3. A Text Area
4. Prediction Scores and choices

The gesture to be recognized is drawn in the Overlay View as shown in Fig. 6 and is reflected persistently in the Canvas area. This ensures that a word is visible in its entirety instead of only the letter that is currently being drawn onscreen.

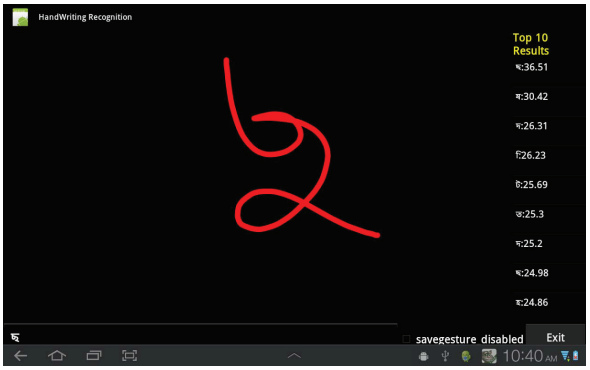

Figure 6. A drawn gesture corresponding to the Bangla character 'Chha' being recognized correctly and its Unicode is placed in the editable Text Area at the bottom of the Tab's screen. The other possible matches are shown in the right sidebar.

For each gesture drawn, we pass it to the recognizer module and the output corresponding to the top 10 nearest character classes (based on prediction scores). What the Gesture Recognizer does is sampling N temporally equidistant points from the strokes of a Gesture object. We thus get vectors ($x_1, y_1, x_2, y_2, \ldots, x_N, y_N$) of equal length for each gesture. The squared Euclidean distance between the input Gesture object and the corresponding points of all the Gesture objects in the library is pair-wise computed. They are assigned scores based on the proximity to the input gesture for a k-nearest neighbor search.

Elastic matching using DTW (dynamic time warp) technique was also studied. However, the flexibility in such recognition was found to be an unwanted quality, especially as there were many competing extra-class samples in the library that were closely similar. A point-to-point correspondence in similarity is preferred in such cases. A similar observation and further details were reported in [14].

A look-up table using a hash map is present of all the class IDs as keys and the Unicode values of the corresponding Bangla character as the data in string format. The top 10 score classes are matched with the hash map to obtain their proper Unicode strings. These are used to display labels of each class and the recognition score in the Prediction Score area.

The top score and its corresponding character class ID are taken as the recognized class. This Unicode string is written into the Text Area. This portion supports the default onscreen keyboard for giving spaces, line break etc.

C. *Auto-learner Module*

A wrong prediction may occur initially, given the wide variation in writing styles among different people and also how these styles often result in close similarity between inter-class characters. In most of such cases, the correct recognition result is present in the top 10 results displayed, differing by a small percentage with the top (incorrect) recognition result. The user can then select the correct one from among the 10 top predictions. The gesture input by the user (as a Gesture object) is then added to the Gesture Library under the name/class ID corresponding to the correct choice as specified by the user. Naturally this new gesture would be a much closer match to any future instances when the user will write this particular character on the touchscreen. Thus after being corrected the system adapts to its user's individual writing style, the accuracy increasing over use.

The result of adding new gestures to the library for each error case may end up in a bloated size. This is avoided by deleting those character samples that are predicted rarely in its class, making space for new user-defined gestures to take its place.

V. Experimental Results and Discussions

The Auto-learner mechanism is effective, as is shown by the increasing trend in percentage of correct recognitions in the chart in Fig. 7. The initial misclassifications are rectified as the system gradually becomes more adapted to the user's handwriting.

For obtaining the experimental results of the proposed system a group of 10 users were considered. Each of them was asked to draw the present characters whose templates are stored in the Gesture Library in 10 different "sessions". In other words, samples from each character class were drawn by each user 10 times. By a "session" we mean the entire list of characters being drawn on the touchscreen once by a user.

The percentage of correct recognitions in the top scored outputs for each session is plotted against each session. A wrong recognition results in correction by the user, in which case the sample gets added to its correct character class. Subsequent sessions result in higher accuracy, ending at the 10th session at 87.8%.

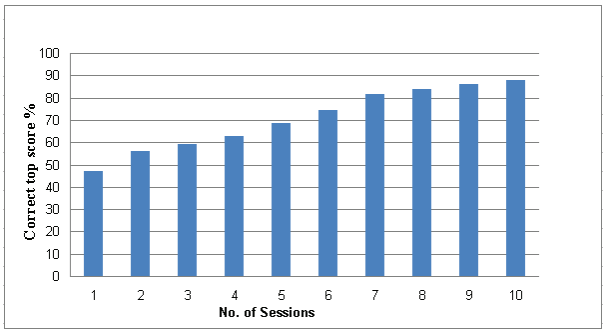

Figure 7. Trends in accuracy of matching and no. of attempts in the personal handwriting recognition system

VI. Conclusion and Future Work

Going back to the three desired features expected of a personal handwriting recognition system, we now compare how our system fares.

1. As the system subsequently stores inputs made by the user, further use of those samples by the user would result in better accuracy as the recognition becomes more customized to the user's own unique writing style.
2. The pre-built gesture library stores samples gathered from a wide range of styles, providing a broad base upon which coarse recognition can initially take place, after which finer tuning is done by incorporating the user's own gestures into that library.
3. Even if incorrect recognition takes place at initial attempts due to lack of a matching template within the concerned character class, user feedback in the form of manual correction over successive usage makes sure that performance improves.

Since we are getting the output of the recognized character as Unicode values and as all Android phones support Unicode character sets, "Intents" can be used to call any desired app such as SMS, note etc. and pass the data (in our case the recognized characters) as the input for those apps. Intents are a feature of Android that provide late runtime bindings amongst different applications and are often used to invoke the services of another Android application in order to perform separate tasks. Thus, there can be easy and seamless integration of this recognizer with other apps.

Further work would be needed to extend this to cursive Bangla handwriting recognition on Android platform and also improve the initial accuracy. Instead of recognizing a sample as a character straightaway, it is possible to first classify each stroke sample into a stroke class and then combining the all the stroke classification results of a character sample, the character classification may be done [15].

ACKNOWLEDGEMENT

This work has been partially supported by the Dept. of Information Technology, Govt. of India.

REFERENCES

[1] M. J. Castro-Bleda S. España J. Gorbe F. Zamora, D. Llorens A. Marzal F. Prat J. M. Vilar, Improving a DTW-based Recognition Engine for On-line Handwritten Characters by Using MLPs, *Proc. of 10th International Conference on Document Analysis and Recognition,* pp. 1260 - 1264, 2009.

[2] R. A. Abdul, M. Khalia, C. Viard-Gaudin, E. Poisson, Online Handwriting Recognition Using Support Vector Machine, *Proc. IEEE Region 10 Conference TENCON 2004,* Vol. 1, pp. 311 – 314, 2004.

[3] M. Nakai, N. Akira, H. Himodaira and S. Sagayama, Substroke Approach to HMM-based On-line Kanji Handwriting Recognition, *Sixth Int. Conf. on Doc. Anal. and Recog. (ICDAR 2001)*, pp. 491-495, 2001.

[4] W. Jiang, Z. Sun. HMM-based On-line Multi-stroke Sketch Recognition. *Proc. of the Fourth International Conference on Machine Learning and Cybernetics, Guangzhou,* 2005, pp.4564-4570, 2005.

[5] J. Sternby, Structurally based template matching of online handwritten characters, in W. Clocksin, A. Fitzgibbon, and P. Torr, (editors), *Proceedings of the 16th British Machine Vision Conference (BMVC '05)*, vol. 1, pp. 250–259, 2005.

[6] Y. Li. Gesture search: a tool for fast mobile data access. *Proc. of the 23nd annual ACM symposium on User interface software and technology*, pp. 87-96, 2010.

[7] T. Y. Ouyang, Y. Li. Bootstrapping Personal Gesture Shortcuts with the Wisdom of the Crowd and Handwriting Recognition. *CHI 2012: ACM Conference on Human Factors in Computing Systems,* Austin, Texas, 2012, pp.2895-2904, 2012.

[8] M. Weber and M. Liwicki, Online Mode Detection: Evaluation on Pen-Enabled Multi-touch Interfaces. *Proc. of International Conference on Document Analysis and Recognition,* pp. 957-961, 2011.

[9] M. Liwicki, O. Rostanin, S. M. El-Neklawy, A. Dengel. Touch & Write: a Multi-Touch Table with Pen-Input. *Proc. of the 9th IAPR International Workshop on Document Analysis Systems,* pp. 479-484, 2010.

[10] T. Mondal, U. Bhattacharya, S. K. Parui, K. Das and D. Mandalapu. On-line handwriting recognition of Indian scripts - the first benchmark. *Proc. of 12th Int. Conf. on Frontiers in Handwriting Recognition,* pp. 200-205, 2010.

[11] U. Bhattacharya, A. Nigam, Y. S. Rawat and S. K. Parui. An analytic scheme for online handwritten Bangla cursive word recognition. *Proc. of the 11th Int. Conf. on Frontiers in Handwriting Recog. (ICFHR 2008)*, pp. 320-325, 2008.

[12] G. A. Fink, S. Vajda, U. Bhattacharya, S.K. Parui and B. B. Chaudhuri. Online Bangla Word Recognition Using Sub-Stroke Level Features and Hidden Markov Models. *Proc. of 12th Int. Conf. on Frontiers in Handwriting Recog. (ICFHR 2010),* pp. 393-398, 2010.

[13] Sk. Mohiuddin, U. Bhattacharya and S. K. Parui. Unconstrained Bangla online handwriting recognition based on MLP and SVM. *Proc. 2011 Joint Workshop on Multilingual OCR and Analytics for Noisy Unstructured Text Data,* ACM Digital Library, Article No.16, 2011.

[14] P. Kristensson and S. Zhai. SHARK2: A large vocabulary shorthand writing system for pen-based computers. *Proc. UIST '04. New York: ACM Press,* pp. 43-52, 2004.

[15] S. K. Parui, K. Guin, U. Bhattacharya, and B. B. Chaudhuri, Online handwritten Bangla character recognition using HMM, Proc. of 19th Int. Conf. on Pattern Recognition, 2008, IEEE Computer Society Press, 2008.

.