5. Development of OHWR System for Malayalam

Anoop M. Namboodiri & Team
IIIT, Hyderabad

Introduction

Recognition of online handwriting is a critical component for enabling computing and in-formation access in Indian languages, primarily because they do not lend well to keyboard based input. In addition, mobile devices, which have become extremely popular in our coun-try, have limited space to accommodate keyboards, which makes pen-based input even more compelling. However, reliable handwriting recognizers are available only for English and a few east asian languages such as Chinese and Korean, and efforts for development of such technologies for Indian languages have been limited in the past.

In this report, we describe out efforts in data collection, annotation, engine development, application integration and performance testing for the language of Malayalam. We concen-trate on the aspects that have been improved from the last review meeting held in July 2012.

About the Scripts

We first provide a quick overview of the script under consideration, its lexical and orthographic structures.

Malayalam is closely related to the languages of Tamil and Sanskrit. The language started as a variant of Tamil that was spoken in regions of Kerala, and evolved its own form, grammar and vocabulary by 500 C.E. [1]. A significant transition from Tamil happened with the introduction of a form of literature called Manipravalam, which freely mixed words from Sanskrit into the language. Most words in traditional Malayalam has its roots in either Tamil or Sanskrit. Due to its lineage to both Sanskrit and Tamil, Malayalam has a large set of alphabets that cover the syllables in both languages.

The influence of Sanskrit has not been limited to the words, but has contributed to the grammar and word formation as well. The agglutinative nature of Sanskrit words in poems have been borrowed by Malayalam and is now used in common speech and writing. This allows one to combine multiple root words into a single word using morphological rules. Its usage is so popular that one often encounters compound words formed from 4 or 5 roots in common use. For example, the words nizhal(shadow), pAva(puppet) and kootthu(play) combine to form a single word: nizhalppavakkootthu. One could even write a whole sentence in a single word or two. The first word in the sentence: mazhayund Ayirunnen gilennavan ArOdANAvO paranjathu is formed from 9 separate root words. Note that this is not just a juxtaposition

of multiple words, but a single word formed by fusing the component words according to the sandhi rules.

Other forms of variation of words in Malayalam are from verb inflections or inflections of subject or object. These variations and compound forms make the modeling of language or creation of dictionaries practically impossible.

The predominant orthographic unit, as in other Indian languages, is a consonant ending in a vowel, referred to as an akshara. The canonical structure is (C)V, where V represents a vowel, and the optional C represents one to three consonants. The system follows the principles of phonology and the writing corresponds to the pronunciation with very few exceptions.

Vowels	അ	ആ	ഇ	ഈ	ഉ	ഊ	ഋ	ഌ		ൠ
	എ	ഏ	ഐ	ഒ	ഓ	ഔ	ം	ഃ		ൡ
Consonants	ക	ഖ	ഗ	ഘ	ങ	ച	ഛ	ജ	ഝ	ഞ
	ട	ഠ	ഡ	ഢ	ണ	ത	ഥ	ദ	ധ	ന
	പ	ഫ	ബ	ഭ	മ	യ	ര	ല	വ	ശ
	ഷ	സ	ഹ	ള	ഴ	റ				
Half-Cons.	ൺ	ൻ	ർ	ൽ	ൾ	ൿ				
Vowel and Consonant Modifier s	ാ	ി	ീ	ു	ൂ	ൃ	ൊ	ോ	ൈ	ൌ
	ോ	ൗ	്	ൢ	ൣ	ൎ	ൄ			
Conjuncts (examples)	ക്ക	ങ്ങ	ച്ച	ഞ്ഞ	ട്ട	ണ്ണ	ത്ത	ശ്യ	ശ്രീ	മ്മ
	ന്റ	റ്റ	ന്ധ	ണ്ഡ	ത്ഭ	ന്ത്ര	ദ്ധ	ജ്ഞ	ഞ്ഛ	ത്രു
Numerals	൦	൧	൨	൩	൪	൫	൬	൭	൮	൯
Others	ൟ	൏	൐	ൔ	ൕ	ൖ	ൗ			

Table 1. Aksharamala: The basic set consists of 16 vowels and 36 consonants.

Table 1 shows the basic vowel and consonant symbols. The two vowels at the top-right of the table are used for writing Sanskrit words only. The script also contains symbols for half-consonants, known as chillu, and symbols for vowels, when they appear together with consonants (consonant modifiers in Table 1). The complexity of the script, in terms of recognition, primarily comes from the similarity between different symbols as well as the large number of symbols that are present in the script. In addition to the above symbols (around 75), there is a very large set of symbols for specific

Old	കൃ	ക്ഷ	ക്ര	ക്യ	വൃ	വൂ	വ്ര	വ്യ
Revised	കൃ	കൃ	ക്ര	ക്യ	വൃ	വൂ	വ്ര	വ്യ
syllable	ku	kuu	kra	kRu	vu	vuu	vra	vRu
Old	ദ്ധ	ശ്ച	ത്മ	ക്ഷ	ക്ത്ര	സ്ക്രൂ	പ്പ	ഞ്ജ
Revised	ദ്ധ	ശ്ച	ത്മ	ക്ഷ	ക്ത്ര	സ്ക്രൂ	പ്പ	ഞ്ജ
syllable	da+dha	sha+cha	ta+ma	ka+Sha	ka+ta+ra	sa+ka+ra+uu	pa+pa+u	nja+ja+u

Table 2. Script Revision: Modified consonants and conjunct characters in both old and new lipis.

combinations of (C)V or conjunct characters, as mentioned before. A small set of conjunct characters are shown in Table 1. The last two rows of Table 1 shows symbols that are present in the script, but not in common use anymore. Arabic numerals are used in practice instead of the script-specific numeral symbols.

The last row shows special symbols for units of ten, hundred, thousand, quarter, half, and three-quarters, and for denoting dates.

The most frequent consonants in the language are ka, ra, la and va, and the most frequent independent vowel is a.

Script Revision: To deal the large number of symbols for printing Malayalam, a simplified or reformed version of the script (lipi) was introduced in 1971. The reform aimed to reduce the complexity of two particular aspects of Malayalam script. First, it recommended the re-placement of a large number of irregular ligatures by a predictable sequence of basic glyphs. Specifically, this involved single consonant syllables ending in u, uu, Ru and those combined with the consonant ra. It also recommended the conversion of complex CCV and CCCV combinations to the corresponding sequence of consonant symbols, joined by the vowel suppressor. Table 2 shows examples of both modifications. Thus, the new lipi of Malayalam is supposed to use consonants and diacritics separately, rather than as complex combined symbols. This new script reduces the number of distinct letters for typeset from around 900 to 90. While it has had some effect on daily practice, this reform has only partially changed the well-established traditional approach, especially in writing. By the arrival of modern word-processors, which removed the limitations on the number and complexity of glyphs, many of the old lipi characters re-emerged in print. However, there is no common agreement on the set of conjunct characters among different fonts. As a result, the modern script is often a mixture of traditional and simplified characters. Even in the same book, one could find the same compound character written in both old and new lipi.

To deal with these we have included all the simplified characters as well as the compound characters enabled by modern word

processors. These cover all the characters that are used in practice by writers trained in the olden and modern schools of writing.

Data Collection and Annotation

Initially we had collected over 100, 000 words from around 700 writers and annotated them at word level. In the last 7 months, we have done a significant amount of annotation at the stroke level. In addition, we have collected sentence or page level data as well as data from mobile devices with capacitive touch interfaces.

Mobile Data Collection

We have collected 15 samples of each of the stroke classes using the mobile data collection interface shown in Figure 1. As the interface collects character level data, the input is automatically annotated with its label shown at the top of the interface.

Fig 1. A screenshot of the Malayalam mobile data collection application showing the Stroke class `gda' and the trace written.

Sentence Level Data

A second problem was to collect and annotate 100 pages of sentence level data. Towards this, we have collected 30 pages of data using Genius note device. The remaining data collection is expected to be over in another month. The data would then be annotated at the word level by alignment of unicode text which has already been created corresponding to each page of data collected.

Stroke Level Annotation of Word Data

Initially , a set of words to be written by each writer, such that it covers all possible strokes that could be written in the complete script were decided. Malayalam is a syllabic language. Similar to most languages of India, each symbol in this language represents a complete syllable. Malayalam has 18 vowels, 36 consonants, and 5 half-consonants. Our word list consisted 120 words. Data was collected on A4 size papers with 30 words per page, with sufficient spacing to make the segmentation of the training data easy.

For the data collection task, we had decided to stick with people who write in the new style of writing, which had become popular around 30 years ago. We had collected word data from around 700 people in Malayalam, amounting to a total of 100, 000 words and characters.

Devices for Data Collection

The primary considerations in deciding on a device for data collection were: i) Cost and ruggedness of the device considering both the ease of data collection, as well as the usefulness of any recognition engine that is developed, ii) The quality of data that is generated by the device, which should be

sufficient for accurate recognition to be performed, and iii) The data format, which should be made available by the device manufacturer, allowing the applications to read the utilize the data. We had tested the data that was collected from four different devices: a) Tablet PC, b) Wacom USB tablet, c) i-ball takenote, and d) Genius notepad. The majority of the data was collected using the Genius device as it was the least expensive and rugged enough to be taken to the field for data collection. The 'TOP' data format used by the device was also accessible due to the reading routines supplied by the device manufacturer.

Annotation Process

The first stage of the annotation process in such a multi-institution process was to come up with a representation standard, which would be followed by all the institutions so that the resulting data is easily usable by anyone, in a uniform manner. Towards this end, the consortium has drafted an XML based data and annotation representation format [2], which has been refined over multiple iterations. All data that we collected was converted to the above XML format during the process of annotation. We carried out the annotation process in the following steps:

1. In the first step, the data collected from the field was inspected to remove any pages corrupted by the device malfunction (mostly due to pen sticking or failing batteries). The data was then separated into separate folders, each corresponding to a specific writer
2. The second step reads the TOP files from each writer, segments it into words, which is manually verified and corrected, and each word is assigned a Unicode label of its content. The results are stored in the XML format mentioned before.
3. In the third stage, a set of strokes from the data are extracted from words with same annotation and are annotated at the stroke level using a clustering and verification process, described below. The annotated strokes are used to train and test the recognition engine.
4. The stroke level annotation is used to carry out segmentation and annotation of aksharas in a semi-automatic fashion.

Word-level Annotation: We have developed a stand-alone annotation tool (designed and developed in QT/C++) for the purpose of word annotation. The tool has undergone multiple revisions and currently can take configuration files for a language and the data collection format, and annotate the data for a user, automatically. The tool also displays the resulting annotation and allows the user to make modifications to the segmentation as well as word labels. It can read files in the TOP format, generated by Genius tablets, and write the annotated results in the XML format described before.

The tool has been shared among the consortium partners, and some of the partners have been able to use it for their annotation work.

Stroke-level Annotation: As mentioned before the annotation of the strokes are carried out by a semi-automatic tool that clusters the strokes and allows for modification and labeling of the clusters or strokes. The process of Stroke Annotation is carried out through the following steps:

1. Stroke Extraction: Collect the stroke from samples of a single character. Take all the

instances of a given character and find out all the strokes present in it.

2. Stroke Clustering: Hierarchical clustering is used. The number of clusters is to be given in toolkit. The elastic matching distance between the two strokes is used as the distance measure for clustering.
3. Cluster/Stoke annotation: The tool allows for modification of the clustering at a fine grained level, and allows quick annotation of a large set of strokes.
4. The tool can also generate the character to stroke dictionaries at this point, which is not utilized at the moment.

Initially, we had annotated around 60 samples per class for the 90 strokes most popular aksharas and 10 digits. We have now extended this to 107 strokes by including most of the compound characters, in addition to the 10 digits and 13 special characters or symbols. This results in a total of 130 classes. The text-only recognizer deals with 107 classes and the complete classifier deals with 130 classes.

We have extended the training data to around 90 samples of each of the 130 classes till now. The accuracy reports are based on this larger dataset. In the next step, we are carrying out a significant increase in stroke-level annotation by collection over 1000 samples of each of the text classes. The process of annotating these strokes is ongoing as per the process described below.

Semi-automated Stroke Level Annotation: In-order to speed up the process of stroke level annotation task, we used a semi-automated method for labeling each stroke trace. For all the 120 words collected in the data collection phase, all the possible stroke classes possible for each word were listed out and stored.

After the stroke level annotation and separation done in the previous step, the strokes corresponding to each class were segregated and an SVM Classifier was designed with the ability to output the probability (confidence) score of each classification. Since the word level annotation task was already done, and each the all possible strokes for each word are known, each trace in a word is classified into one of the possible classes by taking the class with the maximum classification confidence score. Using this method, the entire dataset was annotated with very minimal human intervention.

For the purpose of training our classifier, we need high quality annotations and the semi-automatic method described above is not sufficient. To achieve such high quality annotation, we manually remove any stroke that does not fit the required appearance of the stroke using an offline representation of the stroke. The remaining strokes were collected and used as training data for the classifier. We used this method to initially extract 90 samples of each class, which were used for training our current recognition engine. Later this was extending to around 1000 samples for each stroke class.

Design of Recognition Engine

The overall design of the classifier has not changed, while a lot of finer details in the process have changed. In order to improve the overall accuracy of the classifier, we have experimented with more robust ways to extract features and the results on combining it with existing features.

Feature Extraction

Three new feature extraction methods have been developed to improve the overall accuracy of the system as described below:

Inverse Curvature re-sampling: The popular traditional representations are equitime sampling and equidistant sampling [3]. It has been shown, experimentally, that equidistant resampling has a better discriminative feature space than equitime resampling [4, 3]. One of the problems of using equidistant resampling for recognition of Indic languages is that it, on certain occasions, minor discriminative details such as Cusps vs. small loops are lost. Most of these smaller details happen at areas of high curvature. Thus we also experiment with an alternative method for sampling that is based on curvature.

The curvature is calculated from the spline representation as:

$$\kappa(t) = \left| \frac{\dot{x}\ddot{y} - \ddot{x}\dot{y}}{\sqrt{(\dot{x}^2 + \dot{y}^2)^3}} \right|,$$

where x(t) & y(t) are the x and y trajectories with time. The cumulative curvature is calculated as:

$$h(t) = \int_0^t \kappa(t)\, dt$$

The inverse of h(t) is computed and is evaluated at linearly spaced points in the range [0, max{h(t)}]. This gives the time-intervals {ti} at which the handwriting has to be resampled. The original handwriting is then interpolated at {ti} to get a resampled version in accordance with the curvature of handwriting. We found that, curvature weighted sampling when clubbed with equidistant resampling performs better than either of them individually.

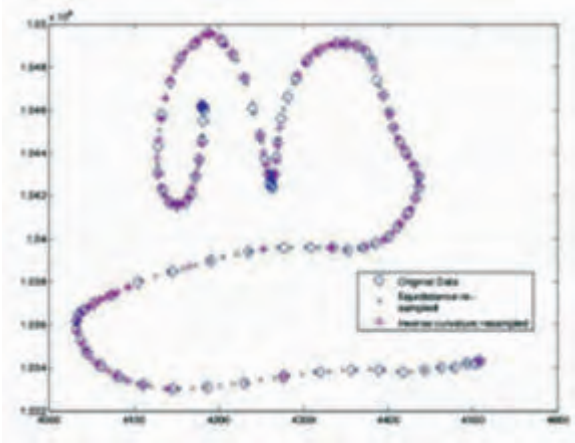

Fig 2. An example of various resampling methods. It is clear that curvature weighted sampling does a better job of preserving areas of higher curvature

Circle based representation of strokes: A ballistic stroke is defined as one movement of hand, which has an asymmetric Gaussian speed curve with a chance of inversion in direction at the starting or ending of the stroke. Traditional method for ballistic stroke segmentation divides a trace at instants of time when speed profile reaches a local minimum. A ballistic stroke, spatially, can be described as a pivotal movement of the hand along the arc of a circle [5]. In order to spatially characterize a stroke, the center, radius of the circle, the starting and ending points of the arc are required. To each of the strokes calculated from the curvature profile, a circle is fit by minimizing the sum of squared radial deviations.

$$\min_{x_0, y_0, r} \sum_{i}^{m} x_i^2 + y_i^2 - 2x_0 x_i - 2y_0 y_i + x_0^2 + y_0^2 + r^2$$

Let $x_0 = a_1$, $y_0 = a_2$, $r = a_3$. For $i = 1 \ldots m$, the above equation can be represented in matrix form as,

$$X\ Y\ 1)\ (a_1\ a_2\ a_3)T = -(X \circ X + Y \circ Y) \quad (2)$$

where $A \circ B$ denotes the Hadamard product of two matrices A & B, X is a column vector of x co-ordinates and Y is a column vector of y co-ordinates of the points to which a circle has to be fit, 1 is a column vector of 1 of length m. The least-squares fit to the system of 2 can be solved by:

$$(a_1\ a_2\ a_3)^T = -(X\ Y\ 1)^+ \times (X \circ X + Y \circ Y) \quad (3)$$

where A^+ is the Moore-Penrose pseudoinverse of A. The center and the radius of the circle can be computed from 3 by

$$x_0 = a_1/2,\ \ y_0 = a_2/2,\quad r = \sqrt{\frac{a_1^2+a_2^2}{4} - a_3}$$

An equivalent of describing the circular arc with the starting and ending points is taking the angles of the lines joining the center of the circle and two ends of the stroke with x-axis. Thus the representation $(x_0, y_0, r, \theta_s, \theta_e)$ describes completely a stroke in space, where θ_s, θ_e denote the angles at the starting and ending points of the stroke.

Each trace is normalized in space to fit into a unit square. The sensitivity of the circle parameters (r, x_0, y_0) to curvature of the arc being fit is high. i.e., the values of (r, x_0, y_0) become exponentially large as the the arc approaches a straight line. In order to mitigate this problem, a monotonic function (say hyperbolic tangent) is used to map the values from R to [0, 1]. Also, it is intuitive that the variation of the mean of a stroke is less compared to the variations in the center of the circle. So we substitute the mean coordinates of the ballistic stroke in the place of center (x_0, y_0), which we will continue to denote the mean coordinate rather than the center of the fitted circle. The hyperbolic function tanh is applied only to the radius dimension, as it is the only unbounded ordinate in the representation.

To apply the above model for character recognition, for all the characters in the training set, the strokes are found out and their circle fits are calculated. The circular features i.e., $(r, x_0, y_0, \theta_s, \theta_e)$ of all the strokes are clustered using k-means into k clusters and the centroids of the clusters are stored. Each character is represented as a Bag-of-Words (BoW) model, with the words defined as these centroids.

In the traditional Bag-of-Words (BoW) model, each document is represented as a histogram of unordered set of words present in that document, ignoring the semantics of it. The feature representation for each character is a k-bit feature vector corresponding to k centroids, which is initialized to zero. For each character's strokes, the circular representation is found. The distance to each of the centroids is calculated and the bit corresponding to the centroid that is closest to the given stroke is set. Thus the entire character is encoded as a string of k-bits. Our representation differs from the traditional Bag-of-Words in such a way that indicator functionality is described rather than a histogram, and we refer to the representation as Bag-of-Strokes. An example of this process for the Malayalam vowel symbol au is illustrated in 3.

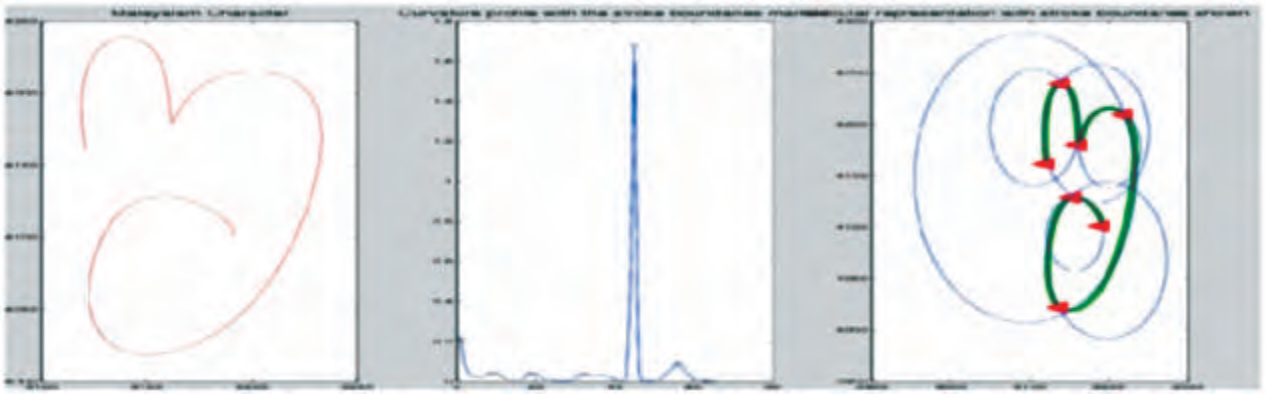

Fig. 3. Example of segmentation and representation.

	Equidistant Sampling(ED)	Curvature weighted sampling(CS)	ED+CS	Bag of Strokes repre Sentation	ED+CS+BoS
Malayalam	84.40	81.75	85.76	94.55	97.75
UJIPenchars	82.51	76.05	86.70	95.8	96.5

Table 3. Recognition accuracies on of various datasets with different feature extraction techniques.

Histogram of Oriented Gradients (HOG): Histogram of Oriented Gradients (HOG) [6] is a famous feature descriptor used in computer vision originally for human detection but now has extensive usage in object detection etc. The core governing idea of HOG is that local object appearance and shape within an image can be described by the distribution of intensity gradients or edge directions without precise knowledge of the corresponding gradient or edge positions. This accomplished by splitting the given image region into smaller cells and accumulating 1-D histogram of local edge orientation which is computed by an image convolution with the Prewitt operator. These smaller windows are taken to be overlapping. The edge detection operation gives the orientation of edge at a given pixel, and histogram bins evenly distributed over 0o − 360o are used to represent a given cell. The concatenation of histograms of all such cells in an image is the HOG feature of that image.

In our experiments, we converted the online stroke into 28×28 offline image. Morphological thickening was used to make the stroke look brighter. The image was divided into 5 × 5 cells which 9 histogram bins for each cell.

On a 90 class dataset, we were able to achieve an accuracy of 90.5385%. When combined with equidistant sampling and curvature weighted sampling, the accuracy increased to 96.3%.

Classifier Design and Comparison

We have tried comparing results based on the above features using the SVM-DDAG

clas-sifier model developed initially with some of the promising new directions in character recognition.

Convolutional Neural Networks: Convolutional neural networks or ConvNets are a multi-stage neural network that try to model the way brain processes visual data. Convolutional neural networks were proposed to ensure some degree of shift, scale and distortion invariance by using local receptive fields, shared weights and spatial sub-sampling. They have been used with great success for optical character recognition. [7]

Lenet-5 comprises 7 layers which contain trainable parameters. Input is a 32 × 32 image. Layer C1 is a convolutional layer with 6 feature maps with each of them connected to a 5 × 5 neighborhood in the input. Layer S2 is a sub-sampling layer, with 6 feature maps of size 14 × 14. Each unit in feature map is connected to a 2 × 2 neighborhood in C1. The four inputs to each unit in S2 are weighted and are passed through a sigmoid function. Layer C3 is convolutional layer with 16 feature maps with 5 × 5 connections. The layers S2 and C3 are not fully connected to derive complementary features and to avoid symmetry. Layer S4 is a sub-sampling layer with 16 feature maps of size 5times5. Layer C5 is a fully connected convolutional layer with 120 feature maps. Layer F6 and the output are from a typical multi-layer perceptron model. The whole network is trained using a back-propagation algorithm.

In our implementation we used a GPU based implementation of CNN. This is resulted in a high speed up in the training time (a few hundred times speed up). We implemented this on the publicly available MNIST database and were able to achieve state-of-the art accuracies of 99.1%. In comparison the SVM based classifier working on HOG features achieved an accruacy of 99.4% on the online digit dataset that we were using. While the numbers are not directly comparable, the results indicate that SVM classifiers using HOG features can perform close to the state of the art results.

Post-processing

As per the design of the classifier, the recognized results are converted to a unicode sequence before output. The post processing routines are defined over this unicode output to keep the recognition and post processing modules independent. This reduces the

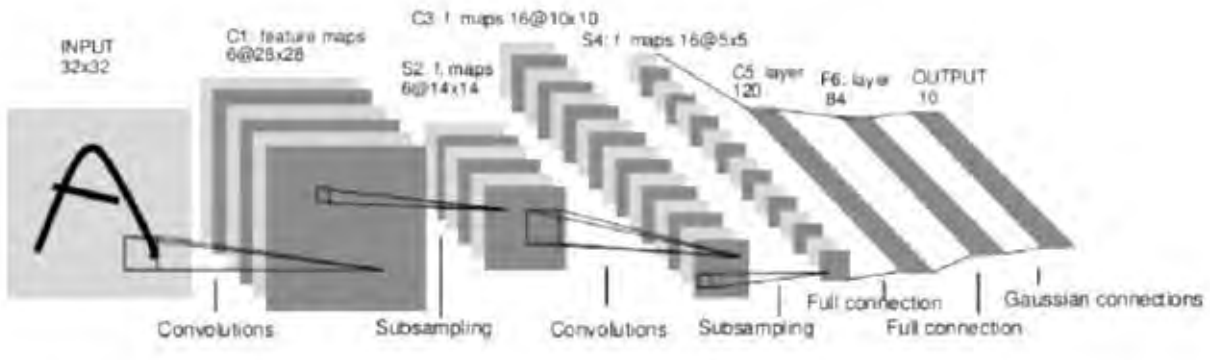

Fig. 4. Architecture of Lenet-5 Convolutional Neural Network[7]

complexity of the design and increases the ability to interchange different recognition and post-processing modules as per requirement of the application.

The current set of post processing routines include dictionary based lookup for fields with a limited dictionary and bigram based post-processing for unrestricted data entry. The for-mer approach includes finding the best match for the recognition output from the dictionary for a specific field such as name of the state or occupation. This involves the computation of edit distance between the unicode output and each of the dictionary words. As the pro-cess is straight forward, we avoid any description here. However, the generation of bigram probabilities and its use are slightly more involved.

Learning Unicode Bigrams

Use of bigram statistics have proven to be an effective way to disambiguate in a variety of real-world scenarios. The power of bigram comes from two different aspects: 1) The ability to score more likely character pairs over those that are less likely, and 2) The ability to avoid character pairs that does not occur in a language.

The effect of the first aspect is self evident, when used in a Bayesian framework. However, determining character pairs that does not occur in a language is tricky. In fact pairs of characters that does not occur in a training text corpus will be assigned zero probability by the training algorithm, where the character pair is in fact allowed in the language. This directly leads to the post processor changing correct recognition results to incorrect ones. To avoid this, we apply regularization to the learned bigram statistics. This process involves assigning a small (non-zero) probability to each pair of characters that were not encountered in the training corpus. However, this would negate the ability of the post processor to avoid pairs of characters that are disallowed in a language.

To get over this problem, we did bigram learning as a two-stage process. In the first stage, a language expert identifies pairs of characters that are allowed and disallowed in the language and creates a binary bigram table. This is done in a spreadsheet for convenience and is exported to a CSV file. A script is used to crawl a website and download html data and extract words in the language of interest. The data is used to generate bigram statistics at unicode level. The script also keeps links to words that include each character pair that is assigned a non-zero probability.

In the second stage regularization is applied to the learned bigram while keeping the pairs that are deemed disallowed by the expert as zero. In any case where a disallowed pair of characters/unicodes appear in the learned table, the script will output the cases with examples of words in which they occur. This can be used by the expert to update the binary bigram table. Once the expert is satisfied with the merge, the resulting bigram table is used for post processing.

Summary and Ongoing Work

We have made significant progress in improving the accuracy and coverage of our recognition engine in the past year and we are ready to go ahead with rigourous testing and fine tuning of the resulting engine. The ongoing work includes incorporating the bigram based post pro-cessing into the

recognition engine and carrying out extensive testing on individual interfaces as well as on the annotated data.

References

1. B. R. Caldwell, Comparative Grammar of Dravidian Languages. Madrass Press, 1875.
2. S. Belhe, S. Chakravarthy, and A. Ramakrishnan, "XML standard for indic online handwritten database," in Proceedings of the Workshop on Multilingual OCRs (ICDAR'09), (Barcelona, Spain), July 2009.
3. R. H. Kassel, A comparison of approaches to on-line handwritten character recognition. PhD thesis, Mas-sachusetts Institute of Technology, 1995.
4. A. Arora and A. Namboodiri, "A hybrid model for recognition of online handwriting in indian scripts," in Frontiers in Handwriting Recognition (ICFHR), 2010 International Conference on, pp. 433–438, IEEE, 2010.
5. C. OReilly and R. Plamondon, "Development of a sigmalognormal representation for on-line signatures," Pattern Recognition, vol. 42, pp. 3324–3337, Dec. 2009.
6. N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on, vol. 1, pp. 886–893, IEEE, 2005.
7. Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recogni-tion," Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, 1998.

❋ ❋ ❋