A Post-Processing Scheme for Malayalam using Statistical Sub-character Language Models

Karthika Mohan
Centre for Visual Information Technology
IIIT-Hyderabad, India
karthika.nair@gmail.com

C. V. Jawahar
Centre for Visual Information Technology
IIIT-Hyderabad, India
jawahar@iiit.ac.in

ABSTRACT

Most of the Indian scripts do not have any robust commercial OCRs. Many of the laboratory prototypes report reasonable results at recognition/classification stage. However, word level accuracies are still poor. It is well known that word accuracy decreases as the number of characters in a word increase. For Malayalam, the average number of characters in a word is almost twice that of English. Moreover, the number of words required to cover 80% of the Malayalam language is more than forty times that of other Indian languages such as Hindi. Hence a direct dictionary based post-processing scheme is not suitable for Malayalam.

In this paper, we propose a post-processing scheme which uses statistical language models at the sub-character level to boost word level recognition results. We use a multi-stage graph representation and formulate the recognition task as an optimization problem. Edges of the graph encode the language information and nodes represent the visual similarities. An optimal path from source node to destination node represents the recognized text. We validate our method on more than 10,000 words from a Malayalam corpus.

1. INTRODUCTION

Robust and accurate OCRs are not yet available for Indian scripts [6, 15]. Recent research has successfully demonstrated solutions to many modules related to segmentation and character classification. There are many laboratory prototypes evolving [6]. However, the recognition accuracies on real-life document collections, still need significant improvement for these OCRs to be applicable in practical situations. Specifically, accuracies at the word and sentence level are low. This can be partly attributed to the challenges in accurate segmentation of document images. Another reason is the difficulty in designing an appropriate post-processing scheme for classifier outputs. For Malayalam, one of the prominent south Indian languages, there is no complete OCR system reported till now [6]. Initial results on classification of individual symbols/characters are promising [13, 14], However, further work is required to enhance the accuracies by exploiting larger context in recognition. This paper investigates a post-processor architecture which suits Malayalam. Though, our results are primarily on Malayalam, our method is also applicable for many other Indic scripts.

DAS '10, June 9-11, 2010, Boston, MA, USA

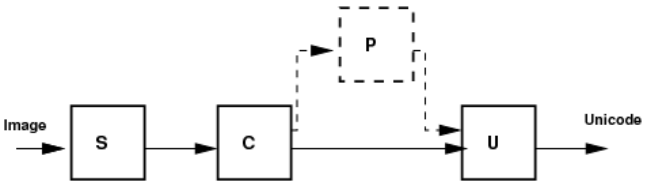


Figure 1: The popular architecture followed in Indian language OCRs. Input image is first (i) S: Segmented and parsed (ii) C:Classified/recognized, and finally (iii) U: converted into Unicode (or any publicly accepted standard)A Post-Processor P closely connected to the classifier could boost word level accuracy.

Multiple studies have attempted to describe the challenges in the design and implementation of Indian language OCRs [6, 15]. Some of these challenges can be summarized as: (i) The incompatibility between the recognition stage, typically at the sub-character level, and the final representation (such as UNICODE[1]). One corollary of this incompatibility is also that different modules of an OCR system aim to achieve different objectives. (ii) The shape and spatial organization of the glyphs in Indic languages create highly complex scripts, which in turn complicate segmentation and parsing. (iii) The presence of conjunct characters indicates the need to solve a large multiclass classification problem. Significantly compounding this is the high probability of confusing similarities between any given pair of classes.

Lack of appropriate statistical models and vocabularies, combined with an inflexional nature of the language make many of the traditional post-processing schemes unsuitable. *A language model computed at Unicode level may not be the natural choice for enhancing the accuracies during post-processing.* Dictionary based post-processing is also not ideal for Malayalam. This is because to obtain around 80% coverage, one

[1]Throughout this paper, we use UNICODE as a representative of publicly acceptable encoding or representation scheme

may require approximately forty times the number of words that is required for Hindi [3]. In fact, this could be in even more since the corpus used for the above study may be limited. For Malayalam, this explosion in number of words can be attributed to (i) complex word morphology and inflexional nature. (ii) large number of foreign words which the language continuously absorbs and (iii) strong and simultaneous influence of *Sanskrit* and *Dravidian* linguistic traditions. We explore a statistical model (at sub-character level) as a possible solution in this paper.

A typical OCR architecture applicable to many of the Indian languages is shown in Figure 1. Almost all of the Indian language OCRs to date (barring exceptions such as [12]) fall into this bottom-up serial architecture. The post-processor operates on the output from the classifier (class labels) and passes the result (again, in the form of class labels) to the Symbol to Unicode conversion module. Primary objective of our work is to design, implement and validate a post-processing module *which fits with such an architecture* thereby enhancing the word level accuracies for Malayalam. This requires computation of linguistic information at sub-character (from now onwards, also referred to as symbol) level. We use this along with classifier scores and visual similarities to come up with an optimal solution.

We now summarize the flow of information in Figure 1, and expose some of the specific issues. The segmentation unit splits the page into words and further down into symbols. This stage is often prone to failure. For some of the Indian languages, this step also involves removal of a connecting headline (*sirorekha*). Segmentation and parsing yield a sequence of (often isolated) symbols. Since the Indic scripts use 2-Dimensional distribution of symbols, parsers are also required to provide spatial information associated with the symbols. They are then recognized using appropriate features and classifiers. Multiple such symbols then combine to generate a character/code. Thus, though the final output is in Unicode, the intermediate modules "optimize" performance at a different resolution (at symbol-level). Note that many sequences of class labels (corresponding to symbols) cannot be converted into valid words in the language or even valid Unicode sequences. This is due to the fact that features and classifiers operate at a different level compared to what we would like the OCRs to do. This sub-character representation makes the detection (segmentation), recognition as well as post-processing very challenging.

The semantically meaningful representation of the output is at the level of *akshara*, which is a combination of consonants and vowels. A word may have only few aksharas. However, it will have typically large number of codes (characters) and even larger number of symbols. Note that the typical number of characters in an English word is around 4 while the number of symbols in Malayalam is 10. Even if we design a reasonably high accuracy classifier at the symbol level, the code, *akshara* and word level accuracies will continue to be low. Consider a situation where there are 10 symbols per word, and the symbol level classification accuracy is 98%. The expected word level accuracy is only 81.7% (i.e., $(0.98)^{10}$). This severely limits the utility of OCRs. For human consumption of the OCR results, we require meaningful *aksharas* and words. This requires the use of higher level knowledge (primarily language model) for enhancing the accuracy. There are also attempts to enhance the accuracy without using an explicit language model [17] for languages where traditional post-processing is challenging. However, they are applicable only when one simultaneously recognizes a larger collection.

Post-processors have been extensively used in the past to enhance OCR accuracies. Various OCR post-processing techniques are available in literature [16, 20]. Popular methods include (i) use of dictionaries or lexicons (ii) syntactic or semantic rules of the language/word morphology (iii) statistical language models usually represented as n-gram etc. Some of the methods find the best alternative, while other methods find the top-n possible alternatives.

There have also been attempts to use a post-processor for enhancing accuracies [2, 11, 15] in Indian scripts. In Bansal *et al* [1], a post-processing scheme for Hindi OCR makes use of a dictionary that is appropriately partitioned. If the input word is found in the partition, it is assumed to be correct otherwise aliases are generated for the word. This technique is not applicable for some other Indian languages (like Telugu, Malayalam) due to the sheer size of their vocabulary. Sharma *et al* [18] have put forth shape encoding based post-processing for Gurumukhi (an Indian script). The consonants are grouped based on shape similarity and a number is assigned to each group or subset. The input word is suitably encoded. If the code exists in the dictionary then match operation is performed between the input word and the words stored under the particular code. If there is no match, then words that are structurally closest to the input word are offered as alternatives. The accuracy of Telugu language OCR has been shown to improve by fusing results of word clusters that are created using Locality Sensitive Hashing(LSH) [17]. The OCR output of word images in a cluster are compared with OCR output of other word images in the same cluster. Recognition results are improved using Character Majority Voting or Dynamic Time Warping Technique.

Most of the previous methods in OCR post-processors employ post-processing at the Unicode (or character) level. We would like to explore the possibility of using a language model based post-processor that is closer to the classification stage. Some of the past work in this area tightly integrate the language model into the recognition/classification stage [10, 12]. Use of statistical noisy channel and language model in post-processing has been discussed in [4]. The post-processing framework comprises of error detection and error correction stages. The detection stage uses image statistical model and the correction stage uses noisy channels and language model. A contextual post processing scheme for Korean OCR has been proposed in [9]. Morphological analyzer is the key component that is used to select the most feasible word from a set of candidate words. Syllable di-grams and viable prefix are used to contain the combinatorial explosion and speed up the system. Confusion matrix corrects the feasible words. However, such methods are not very popular for Indic scripts. Our multistage graph-based solution integrates (i) classifier outputs (ii) language models at symbol-level (iii) visual similarity (iv) validity based on a script grammar, to produce a robust post-processor.

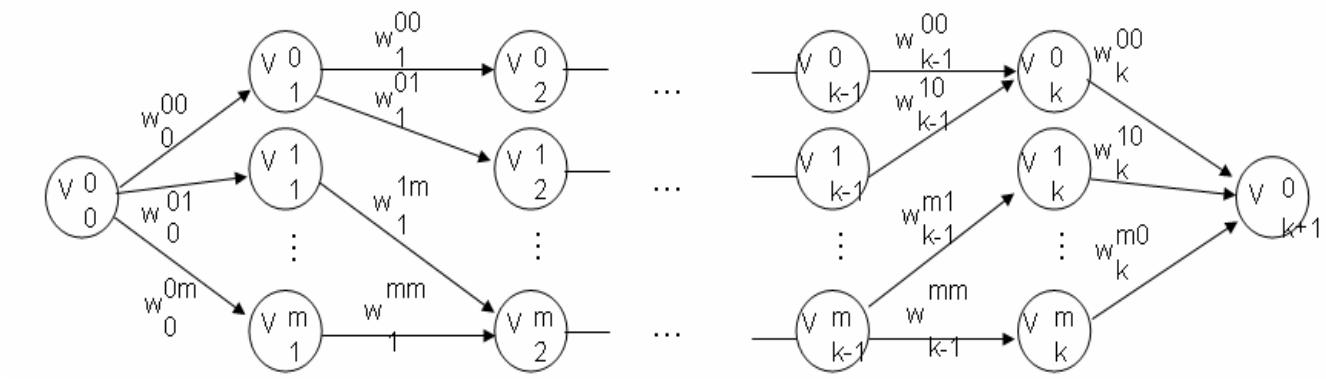


Figure 2: An Example Multistage Graph Representation with Bounded List of Nodes

2. MULTI STAGE GRAPH REPRESENTATION

A post-processor module which fits into a bottom up architecture (as shown in Figure 1) should be able to: (i) Generate alternate words based on the outputs of the previous module. If the classifier had given a ranked list of classes, it would have been trivial to generate alternate possibilities. However, many modern classifiers (like SVMs) do not directly provide probabilities or alternate options. (ii) Use linguistic information to complement the classifier outputs and generate meaningful words. (iii) Provide a mechanism to rank and validate these alternatives and pick the most valid (suited) word. For example, many of the possibilities in the above list need not obey the script grammar. (Note that only certain combination of symbols can correspond to a Unicode or *akshara*) (iv) Confirm the validity of the new word through a validation/verification step and facilitate the replacement of the current output, if required. Since the linguistic resources are still not very rich, this is a very critical decision. However the available linguistic information from dictionary or n-grams can aid this decision making process.

We model this problem as a reasoning (shortest path) problem in a multistage graph. A multistage graph consists of two or more stages such that directed edges exist only from a stage to the next/adjacent stage. An example of the multistage graph, we use, is shown Figure 2. Given the classifier outputs (i.e., sequence of class labels), we would like to generate a ranked list of probable valid words (in Unicode). Using the probabilistic outputs of the classifier, the problem could also have been formulated in a PFSA framework as in [19]. For any given string, the probability of it being generated, by PFAs can be computed by employing a dynamic programming algorithm. However, the process of computing most probable string (i.e. string with the highest probability of generation) is NP-Hard [19]. In our case, we are not only interested in the string with highest probability but also are interested in getting a ranked list of alternate words. Multi Stage Graph (MSG) fits this requirement well. MSG truly represents the problem at hand because the probabilistic relationships from classifier as well as statistical language model can directly map into this structure. The most probable string may be computed using dynamic programming. Efficiency can be improved by using bounding conditions so that solutions that are unlikely to form a part of the desired solution set are rejected instantly. We demonstrate this later.

Construction of Graph. To start with, let us assume that there is no cut or merge in the word image. Therefore each symbol can be detected and ordered correctly. Given a word with k symbols, the MSG given by, $G =< V, E >$ can be constructed as shown in Figure 2. We can partition the nodes in the graph into $k+2$ disjoint sets $\mathbf{V_i}$ $0 \leq i \leq k+1$ where each $\mathbf{V_i}$ defines a stage in the graph. The first and last stages are special stages – source and destination respectively. The node V_i^j is the j^{th} node in i^{th} stage. The k stages of the graph correspond to the k symbols of the word under consideration. The j^{th} node in a stage indicates that symbol is in node j and characterizes the probability that is encoded as the weight P_i^j of the node.

If $(u, v) \in E$, then (u, v) is a directed edge from $u \in \mathbf{V_i}$ to $v \in \mathbf{V_{i+1}}$ such that $1 \leq i \leq k$. The weight of edge (u, v), w_i^{jk} is the probability of occurrence of two symbols u and v consecutively in the word image corpus. Note that these probabilities are similar to the bigrams traditionally used in language modelling. However, here u and v are symbols, and their spatial adjacency in the printed word image is also captured in the probability distribution. We refer to this distribution as statistical sub-character language model (SSLM). All nodes in the first stage are connected to the source node and all nodes in the kth stage are connected to the destination node. The corresponding edges are assigned a cost of one.

Any path from source to destination has a cost defined as the product of node and edge weights on the path, given by $\Pi_{i=0}^{k} P_i^j w_i^{jk}$. This is the probability of generation of the word image. If every stage has 300 nodes (eg: number of classes) and 10 stages, the number of possible paths will be enormous. There exists a dynamic programming based solution which can generate the most optimal path efficiently. However, generation of best-N paths is more involved. First, we restrict our search space by restricting the number of nodes per stage to be a small number (typically less than 10). This reduces the space complexity. We further improve the computation by bounding the paths as discussed later.

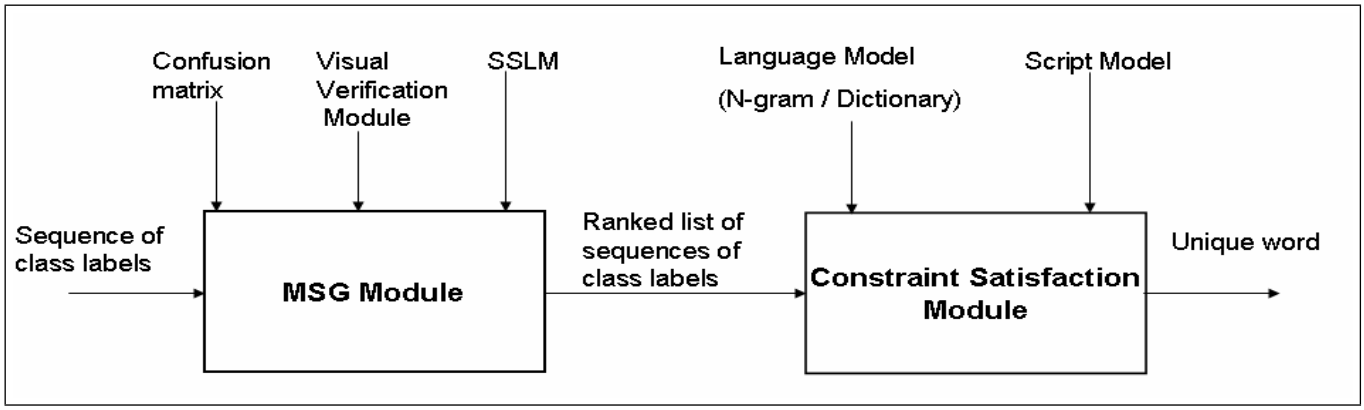


Figure 3: Overview of the Post-Processing Scheme

Some of the classifiers can directly provide the node weight P_i^j, probability of the i^{th} symbol to be in the class corresponding to j^{th} node. If it is not available (as in our case), we use the confusion matrix information to assign the node weights. We augmented this information with a visual verification score (VVS) as we discuss in the next section. Thus the post-processing problem can be formulated as the generation of a valid (according to script and language rules) path with highest probability. Our solution generates top-N highest probability paths, and identifies the best one which meets the script and language constraints.

To compute the SSLM, one could use a corpus of annotated word images (at the symbol level). However, obtaining large such corpus is impractical. We use a reverse script grammar to generate the symbol distribution from the text and compute the model. Even then the estimates could be noisy, due to the sparseness of data. We employ smoothing techniques to ensure that the lack of training data does nor result in assigning zero to valid class label pairs. Various techniques that could be used for smoothing is discussed in detail [5]. There are invalid pairs of symbols which can not co-occur. These probabilities are explicitly made zero.

Generation of Alternate Words. Optimal path in MSG may be computed using dynamic programming [8]. This can be done using forward costs (traversing from the source to the destination) or the backward costs (traversing from destination to source). However, this yields only the optimal word. We are interested in not just the optimal word but a list of top scored words. Our procedure is defined in Algorithm 1. We traverse the graph stage by stage from the source node to the destination node. This results in the creation of partial sequences or patterns. We add them to the solution set. Scores of partial sequences are computed and at the end of the process we normalize the scores of all sequences. Whenever the number of partial sequence in the solution set exceeds a predefined value M, the solution set is pruned. Thus we get top M words that are sorted in descending order, based on their scores.

2.1 Efficiency through Branch and Bound

Even if we prune the number nodes in a stage, the number of possible paths could be still very high. For example, if $k = 10$ and the number of nodes in every stage is eight, then the number of possible paths is $8^{10} = 1,073,741,824$. This is certainly unacceptable. This has a significant impact on time complexity. We capitalize on the branch and bound technique to improve the overall efficiency of the solution. The graph is traversed in such a way that only promising paths are explored. In other words, a partial sequence in the solution set is compared with all nodes in the succeeding stage before the next partial word is operated upon. When the size of the solution set exceeds a predefined limit (here, M), we prune the solution set and eliminate all partial words that are least likely to yield the desired word. The nodes in every stage are also sorted according to their weights. The ratio of weights of adjacent nodes is noted. Whenever there is a significant change in the ratio calculated, we remove the last considered node and the nodes following that, from the particular stage. This way we limit the number of nodes in every stage. During the experiments, we have noticed that the average time taken by the module to generate 30 top ranked words is approximately $4ms$. Total memory requirement for the process falls below 5 MB. It has been observed that even on restricting the maximum number of nodes per stage to 5 and assigning a value of twenty to M, we are able to obtain the desired performance from MSG module for the Malayalam corpus, we considered.

Algorithm 1 Generation of M Alternate Words

Require: MSG, SSLM, M
1: **for** each stage $s = 1, \ldots, k+2$ in MSG **do**
2: **for** each partial word pw in solution **do**
3: $sym1$ = last symbol inserted in pw
4: **for** each symbol $sym2$ in stage s **do**
5: if SSLM($sym1$,$sym2$) $\neq 0$
6: update solution set
7: **end for**
8: if size of solution set exceeds M
9: prune solution set
10: **end for**
11: **end for**

3. POST-PROCESSING

Overview of post-processing scheme is presented in Figure 3. Post-processor uses the available and *apriori* information to make an optimal decision. The post-processor architecture comprises of modules like multistage graph (MSG), Visual

Verification, Dictionary / n-gram. The visual verification module computes a score in the range $[0-1]$ which indicates the validity of the symbol in the specific class of interest. This is different from the posterior probability estimate obtained from the multiclass classifier employed. The MSG module relies mainly on the SSLM to correct errors in potentially incorrect words. Aided by confusion matrix, it takes symbol level output from the classifier and produces a set of ranked words. These words are then processed in succession by a second module. This module determines the consistency of the given alternate words with the script and language structure.

Classifier outputs (class labels) as well as confusion matrices are used to generate alternate words, as discussed in the previous section. We consider the node weights P_i^j as the weighted sum of classifier scores and visual scores.

$$P_i^j = \alpha * Score_{classifier} + (1-\alpha) * Score_{vv}$$

where $\alpha \in [0,1]$. Therefore all the scores range from 0 to 1. The words are sorted (in descending order) based on their score. The top two or more words may have identical scores or negligible difference between scores. We are faced with the task of making the right choice and picking the correct word from a list of highly probable candidates. This could be done interactively with the help of a human. In order to fully automate the system and eliminate the need of a human intervention in the correction process, we introduce the constraint satisfaction module. This module holds the necessary language and script information. Given a word, this module can *verify* whether this is valid or not. Use of dictionary for verification may not be feasible due to reasons mentioned in Section 1. It is acknowledged that memory requirements for a n-gram based module is high. Hence a lower value of n is always preferred over a higher value. The architecture employs multiple modules to process data in a linear manner. The advantage of such an architecture is that at any point new modules can be plugged in easily. As it stands, our implementation of constraint satisfaction module is primitive.

3.1 Discussions

OCR performs a classification operation wherein it places the input into a particular group based on certain characteristics that are specific to the input. The post-processor is generally a verification system that determines if an input belongs to the specified class or not. Clearly, verification is simpler to perform.

Classifiers may not employ all the features available. For example, structural features are not used by our classifier. However a verification module can make use of structural features. So we design the visual verification module in such a way that it employs the features that were not used by the classifier during recognition process. Hong *et al* [7] gives some insight into post-processing based on visual relationships. We also explain the need for an additional module like visual verification, from the implementation point of view.

Use of pure language models alone for the correction of words may not always bring a significant change in word or character accuracy levels. This is because the diagonal elements of the confusion matrix, usually have a higher value when compared to non-diagonal elements. Therefore they need not favor the replacement of a symbol with a new symbol even if the initial symbol is erroneous. To overcome this influence of confusion matrix and to increase the correction rates, we make use of the visual characteristics of the word image. To facilitate this we introduce α as mentioned earlier. When alpha takes a value of one, we do not make use of the visual matching score. This is useful especially in cases where the images are degraded or of low quality. Our visual verification score is computed as the similarity of symbol image with a rendered version, computed using heuristic features.

Degraded documents essentially have plenty of cuts and merges in them. If there are cuts, then the number of symbols produced by OCR would be greater than the number of symbols in the actual word. As part of corrective measures we have to merge the disconnected symbols into one. On the other hand, if there are merges, then the number of symbols produced by OCR will be lesser than the number of symbols in the actual word. To correct merges we have to cut it into respective symbols. We handle cuts and merges in degraded word images by following the principles mentioned in [10]. We proceed in two stages. In the first stage merges are handled. As part of the first step we divide the word image into symbols or separate pieces of ink. Then we create as many cuts as possible in each symbol i.e. we generate a set of candidate cuts. These cuts are suitably combined and sent to the classifier. If the class label returned by the classifier and initial class label of the symbol match, then we conclude that there is no merge. Otherwise we know that there is a merge. We suitably group various cuts in an optimal manner, send them to the classifier and identify the merged symbols. Once every symbol is checked for merges, we proceed to check for cuts. In this step we consider each symbol as a cut and repeat the same process to detect cuts, if any.

Algorithm 2 Symbol level smoothing

Require: N, set of words W
for each word w in W **do**
  Convert w to S, set of symbols $s_1, s_2, s_3 ... s_n$
  Reorder elements in S
  for each symbol s_i in S **do**
    Record occurrences of s_i
    if $i \neq n$, record occurrences of s_i, s_{i+1}
  end for
end for
for each b_i in SSLM **do**
  smooth b_i
end for

4. RESULTS AND DISCUSSIONS

We now present the experimental procedure and results, highlighting the utility and limitation of the proposed method. We start by computing the SSLM as briefly sketched in Algorithm 2. We compute this statistical language model at the symbol-level (sub-character). We smooth the probabilities during the computations [5]. This helps in overcoming problems caused due to data sparseness in the corpus. We use a similar smoothing strategy while computing the confusion matrix too. This ensures that we consider rare confusions between two class labels as well. For languages like

Malayalam, there are no standard and reliable corpus available for the computation of language model. For computing the language model, we use our own corpus. Since there is no standard corpus available, we do compute the language model with the entire available corpus (we call this as SSLM-1), and in a leave-one-out (we call this as SSLM-2) manner. In leave one out framework, the word to be post-processed is discarded from the corpus, if present. Leave one out strategy is used to ensure that the accidental occurrence of the word in the corpus does not explicitly bias the post-processing.

Throughout this section, we show how classification accuracy improves with the post-processing scheme. We consider the classification accuracy ρ as

$$\rho = \sum_{j=1}^{n} C_{jj} / \sum_{j=1}^{n} \sum_{i=1}^{n} C_{ij}$$

where C_{ij} is the confusion of ith and jth class. First we show that when confusions are minimal, the overall accuracy at symbol level can be boosted by the SSLM. This is possible due to the fact that the high visual confusions (present in the confusion matrix) need not be correlated to the high probabilities in SSLM.

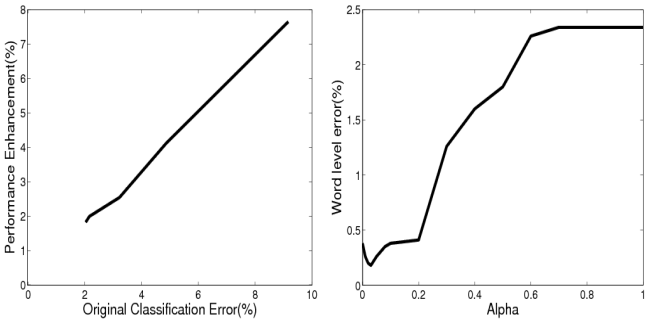
We first demonstrate the utility and limitation of SSLM for enhancing classification accuracy. The confusion matrix 'C' used in the following experiment was created by comparing the symbols generated by the classifier with the ground truth. Consider a sequence of symbols $...s_i s_j...$ where s_i is followed by s_j. Let s_i be classified as z confidently. Assume that s_j is classified as x. However, x frequently gets confused with y. In other words, the entry in the confusion matrix, C_{xy}, is relatively high. This confusion can be disambiguated with the aid of SSLM if P_{zx} and P_{zy} are significantly different. We assign s_j as x if $C_{xx} \cdot P_{zx} > C_{xy} \cdot P_{zy}$. Else we assign x as y. This holds true for all z. Now, we have a new error estimate of C_{xy} denoted as $\hat{C}_{xy}$. It is dependent on the existing confusion matrix C_{xy} and the absolute difference between P_{zx} and P_{zy}. $\hat{C}_{xy}$ is computed for every pair (x, y). Thus the estimate of the accuracy after this post-processing can be estimated as, $\rho' = \sum_{j=1}^{n} \hat{C}_{jj} / \sum_{j=1}^{n} \sum_{i=1}^{n} \hat{C}_{ji}$.

Thus the post-processor enhances overall accuracy and we present the relationship between original classification error: $1 - \rho$ and performance enhancement: $((\rho' - \rho)/\rho) * 100$ for Malayalam script in Figure 4(a). Note that this is an analytically computed graph (and not the results on the corpus) with the help of SSLMs. When the classification accuracy is around 90%, the gain obtained by the post-processing is approximately 8% of initial classification accuracy. As ρ increases, this reduces. This is naturally expected.

Alt	Initial Accuracy		Final Accuracy		
	Char	Word	Char	Word	FN
1	96.42	80.65	98.20	90.19	2.94
2	96.42	80.65	99.28	95.98	0.00

Table 1: Pure language model based correction. Alt: number of alternate words considered, FN: False Negative

The efficacy of SSLM has been confirmed by performing an



(a) (b)

Figure 4: (a) Performance enhancement Vs original classification error(b) Word level Error vs α

experiment on 5000 words. Only language model based correction is employed while processing the words. The initial word level accuracy was 80.65% and character level accuracy was 96.42%. We observed that post-processing significantly increased word and character level accuracy. Results of this experiment is shown in Table 1. The first column 'Alt' indicates the number of alternate words that were considered while computing the word level and character level accuracies.When 'Alt'= n, we consider an erroneous word to be corrected if the correct word is one among the top-n scored words. For example, when 'Alt'=1, we replace all words with the top ranked word resulting in a word accuracy of 90.19% and corresponding character accuracy of 98.2%. This shows the utility of the pure language model, This does not use the visual verification score.

As mentioned in previous sections, probability or score is associated with every word in the list of alternate words generated. In cases where the difference between scores of first alternate word and subsequent words are not significant, we can use a dictionary or n-gram based technique to parse through the remaining alternate words in the result set and choose the correct word. It has been often observed that an initial error rate of 7 percent reduces to 2 percent, 1 percent, 0.8 percent and 0.6 percent when the top 5,10,15 and 20 words are parsed.

The number of character level errors in a word is usually zero or one and sometimes more than one. We say that performance has been deteriorated when errors are introduced into the correctly recognized words. This happens when class labels get replaced by alternate class labels in cases where the corresponding SSLM values are high. The column FN (False Negative) in Table 1 gives an indication of the percentage of correct words that were made incorrect by the system. We curb this tendency by using visual verification module. This module attempts to eliminate incorrect results by offsetting them with visual match scores, which are higher for a correct match between a class and a template. Hence given a word with p errors, either the system will reduce the number of errors at character level or give the same p-error word as output.

Parameter α facilitates the use of visual verification module to calculate the score of a node. Figure 4(b) shows the vari-

ation of error rates with alpha. We observe that error rate is minimum when alpha value is between 0 and 0.1. It may be noted that as part of pruning, nodes in all stages were initially sorted based on weight/probability of occurrence. For most of our experiments we were able to get an optimal performance using α= 0.01. However this may require further verification on a larger corpus.

4.1 Performance on real data

We had seen an improvement in accuracy at word level and character level when the SSLM was used. To facilitate correction of erroneous words, we make use of the visual verification unit. Figure 5 depicts the results on images when only language model(LM)was used, only visual verification(VV) is used and when both were used. We found that though in some cases language model worked and in some VV module worked, the performance was better when both the modules were employed.

Figure 5: Performance of post-processor on real data. LM: Only Language model, VV: Only Visual Verification

Looking at the Figure 5, one can understand the intricacies of Malayalam script. There are symbols that are very similar to each other in appearance. This can be the cause for a great deal of confusion. Often, images have cuts and merges that complicate the recognition process. While post-processing, in some cases, language model corrects the word and in some cases the visual verification unit comes handy. We observe that post-processor performance is optimal when both language model and visual characteristics are used. Weightage of these modules are based on various factors like accuracy of language model at hand, quality of image etc.

Language models are generated from large data sets. In the following experiment we analyze the change in performance or accuracy when SSLM was created from the same corpus and when it was created from a different corpus.We analyze the effect of the data source used to create the language model on the accuracy of the system. This experiment is performed on a data set of 1000 words with initial word error rate of 6.6% and character level error rate of 0.918 %. The results are shown in Table 2.

We observed that word error rate comes down by 3.6% in both cases and the character error rate is slightly lesser when the language model was created from the data source on which it was tested. The deterioration in performance or false negatives is 1.1% in both cases. One can conclude

LM	Initial Accuracy		Final Accuracy		
	Char	Word	Char	Word	FN
SSLM-1	99.08	93.40	99.56	97.00	1.10
SSLM-2	99.08	93.40	99.55	97.00	1.10

Table 2: Effect of language model training set.LM: Language Model,FN: False Negative

that accuracy of language models based correction technique is independent of the training data used to generate the language model. We have a standard language model and use this for all our experiments.

The main focus of our work is to correct errors in the OCR output. Since correct words abide by the language rules, they are unlikely to be affected or deteriorated by the post-processor. Hence in this experiment the data set consists of only erroneous words. We have picked 1000 words that contain one or more errors. The aim of this experiment is to estimate the ability of the language model based post-processor to correct errors in these words. The word level accuracy of input is 0% and the character level accuracy is 86.65%. Results are presented in Table 3.

Alt	Initial Accuracy		Final Accuracy		
	Char	Word	Char	Word	FN
1	86.65	0	98.40	87.15	0.00
5	86.65	0	99.15	93.20	0.00

Table 3: Results on Error Words. Alt: Number of alternate words considered, FN: False Negative

The first column is the number of alternate words considered while computing the accuracy. On comparing the output with the ground truth we see that word level accuracy has risen by 87.15% and character level accuracy has risen by 11.75% when the first alternate word is used. The improvement in accuracy is significant when more number of alternate words are considered. An increase of 87.15% in word level accuracy is not trivial and we may conclude that the post-processor can be used widely for word error correction.

Degraded documents are a major concern and have always been a challenge. Words with cuts like those in the Figure 6 were processed and we were able to combine the cuts and form meaningful words. The techniques discussed in the previous sections were employed to handle cuts. The Figure 6, shows how post-processor handles cuts and generate correct outputs.

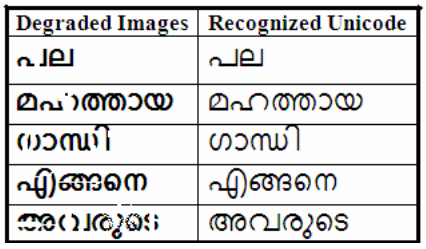


Figure 6: Post-processing of Images with Cuts

The first two inputs have a single cut. In the first case glyphs formed by cuts are close by where as in the second case they are quite apart. The third word image had cuts in two symbols and the fourth has a horizontal cut. The fifth image is a degraded image with several cuts. The results are shown on the right side. We are able to handle degraded documents effectively.

The next experiment is on real corpus of 10,000 words.We take a dataset comprising of correct words and words with errors. The dataset contains 7,000 words with no errors, 2000 words with one error and remaining 1000 words with two or more errors and degradations. The dataset was chosen from different books. The initial word level accuracy is 70%. When fed to the post-processor, the accuracy rate increased significantly. The results are shown in Table 4

Alt	Initial Accuracy		Final Accuracy		
	Char	Word	Char	Word	FN
1	95.31	70.00	98.72	93.42	0.12
5	95.31	70.00	99.19	96.52	0.00

Table 4: Results on Real Data by varying no. of alternate words. Alt: Number of alternate words considered, PP:Post- Processor, FN: False Negative

We observe that a high accuracy of 96% can be attained when words are correctly picked from the first and second highest scored words. The task of choosing the correct word from the list of alternate words was given to the constraint satisfaction module that is aided by n-gram models. When we employed a trigram model we were able to reach an accuracy of 95% at word level. The failure cases included proper nouns and words from foreign languages that contain rare combination of symbols that are assigned a lower score.

5. CONCLUSIONS AND FUTURE WORK

Our approach of using multi stage graph based reasoning, aided by sub-character level language model has proved to be successful in correcting errors in words generated by OCR. We believe that proximity of post-processor to the classification module can significantly improve the robustness of the OCR system. In Indic scripts, classification and post-processing stages are to be tightly coupled. These stages should use the script and language information in the optimization process. We would be conducting large scale experiments on varied malayalam corpora of substantial size. We expect that these experiments would help us in reaching a conclusion regarding the utility, variation and optimal value of threshold parameters such as alpha. Our future work would encompass testing this technique on complex scripts such as Telugu. We intend to enhance the performance and efficiency of the OCR by adopting discrete optimization techniques to optimize recognition and post-processing stages.

6. ACKNOWLEDGEMENTS

This work is supported by Ministry of Communications and Information Technology, Government of India.

7. REFERENCES

[1] V. Bansal and R. Sinha. A complete OCR for printed hindi text in devanagari script. In *Proc. ICDAR*, pages 800–804, 2001.

[2] B.B.Chaudhuri and U. Pal. OCR error detection and correction of an inflectional indian language script. In *Proc. 13th ICPR*, pages 245–249, 1996.

[3] A. Bharati, P. Rao, R. Sangal, and S. M. Bendre. Basic statistical analaysis of corpus and cross comparision. In *Proc. ICON*, 2002.

[4] J. J. S. Chang and S.-D. Chen. The postprocessing of optical character recognition based on statistical noisy channel and language model. *Proceedings of PACLIC*, pages 127–132, 1995.

[5] S. F. Chen and J. Goodman. An empirical study of smoothing techniques for language modeling. *ACM(1996)*, pages 1–3, 1996.

[6] V. Govindaraju and S. Setlur, editors. *Guide to OCR for Indic Scripts*. Springer, Sep 2009.

[7] T. Hong and J. J. Hull. Algorithms for postprocessing ocr results with visual inter-word constraints. *ICIP'95*.

[8] E. Horowitz, S. Sahni, and S. Rajasekaran. *Fundamentals of Computer Algorithms*. Galgotia Publications pvt ltd, 2006.

[9] H.-C. Kwon, H.-J. Hwang, M.-J. Kim, and S.-W. Lee. Contextual postprocessing of a korean ocr system by linguistic constraints. *ICDAR'95*, 2:557–562.

[10] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. of IEEE*, pages 2278–2324, 1998.

[11] G. S. Lehal, C. Singh, and R. Lehal. A shape based post processor for gurumukhi ocr. In *Proc. ICDAR*, 2001.

[12] P. Natarajan, E. MacRostie, , and M. Decerbo. The BBN Byblos Hindi OCR System. *Guide to OCR for Indic Scripts*, pages 173–180, 2009.

[13] N. V. Neeba and C. V. Jawahar. Empirical evaluation of character classification schemes. In *Proc. of ICAPR*, 2009.

[14] N. V. Neeba, A. M. Namboodiri, C. V. jawahar, and P. J. Narayanan. Recognition of Malayalam Documents. *Guide to OCR for Indic Scripts*, pages 125–146, 2009.

[15] U. Pal and B. Chaudhuri. Indian script character recognition: A survey. *Pattern Recognition*, 37(9):1887–1899, 2004.

[16] J. C. Perez-Cortes, J. C. Amengual, J. Arlandis, and R. Llobet. Stochastic error-correcting parsing for OCR post processing. *ICPR'00*, 4:405–408, 2000.

[17] V. Rasagna, A. Kumar, C. V. Jawahar, and R. Manmatha. Robust recognition of documents by fusing results of word clusters. *ICDAR 09*, 2009.

[18] D. V. Sharma, G. S. Lehal, and S. Mehta. Shape encoded post processing of gurmukhi OCR. *ICDAR'09*, pages 788–792, 2009.

[19] E. Vidal, F. Thollard, C. de la Higuera, F. Casacuberta, and R. C. Carrasco. Probabilistic finite state machines- part 1. *IEEE Trans PAMI*, pages 1013–1018, 2005.

[20] L. Zhuang and X. Zhu. An OCR post processing approach based on multi-knowledge. *Knowledge-Based Intelligent Information and Engineering Systems*, pages 346–352, 2005.