12-2022

Character Spotting and Autonomous Tagging: Offline Handwriting Recognition for Bangla, Korean and Other Alphabetic Scripts

Nishatul Majid
Fort Lewis College

Elisa H. Barney Smith
Boise State University

Character Spotting and Autonomous Tagging: Offline Handwriting Recognition for Bangla, Korean and other Alphabetic Scripts

Nishatul Majid
Department of Physics and Engineering,
Fort Lewis College, Durango, Colorado, USA
E-mail: nmajid@fortlewis.edu

Elisa H. Barney Smith
Department of Electrical and Computer Engineering,
Boise State University, Boise, Idaho, USA
E-mail: ebarneysmith@boisestate.edu

Abstract: This paper demonstrates a framework for offline handwriting recognition using character spotting and autonomous tagging which works for any alphabetic script. Character spotting builds on the idea of object detection to find character elements in unsegmented word images. An autonomous tagging approach is introduced which automates the production of a character image training set by estimating character locations in a word based on typical character size. Although scripts can vary vividly from each other, our proposed approach provides a simple and powerful workflow for unconstrained offline recognition that should work for any alphabetic script with few adjustments. Here we demonstrate this approach with handwritten Bangla, obtaining a Character Recognition Accuracy (CRA) of 94.8% and 91.12% with precision and autonomous tagging respectively. Furthermore, we explained how character spotting and autonomous tagging can be implemented for other alphabetic scripts. We demonstrated that with handwritten Hangul/Korean obtaining a Jamo Recognition Accuracy (JRA) of 93.16% using a tiny fraction of the PE92 training set. The combination of character spotting and autonomous tagging takes away one of the biggest frustrations - data annotation by hand; and thus we believe this has the potential to revolutionize the growth of offline recognition development.

Keywords: Offline Handwriting recognition, Bangla Handwriting Recognition, Korean Handwriting Recognition, Character Spotting, Autonomous Tagging

1 Introduction

This paper presents a segmentation free unconstrained offline handwriting recognition framework. This is based on an object detection approach, we call character spotting, that we initially presented in [1] for recognizing Bangla script. In [1] we developed an offline Bangla unconstrained handwriting recognition scheme using sequential detection of characters and diacritics. This paper expands on our original idea to make this framework versatile (i.e. works for most scripts with minor to no modification), easy to achieve (regardless of the nature of the script) and high performing (fast and accurate). The largest bottleneck of our initial approach was the need for character level location data for the neural training. Tagging character locations in handwritten document images is extremely time consuming, and is also an error prone process. Even if the character spotting algorithm works perfectly for a script, this character tagging process makes it very difficult to develop a system. Here

we introduce a process we call "Autonomous Tagging", which removes this step and makes the overall offline recognition development process very approachable for any alphabetic script. Autonomous tagging is a process of loosely estimating a character's location in a word image based on its transcription. We demonstrate the whole development process thoroughly from data acquisition to post processing for the Bangla script. Bangla is a connected script with complexities including vowel diacritics and fused consonants. This is the most versatile and best performing unconstrained offline Bangla recognizer reported to date. We trained our network from the extended Boise State Bangla Handwriting dataset [2], that was carefully crafted so it covers more than 99% of the Bangla script, including all the characters, diacritics and most of the conjuncts. We tested the robustness of our system by testing with three different Bangla datasets. To show the versatility of this approach, we also demonstrate the system on a completely different kind of writing system, the Korean/Hangul script. Hangul is a two-dimensional script, where letters can appear on top of each other as well as along sides. For Hangul recognition, we trained using a fraction of the handwritten Hangul PE92 dataset, and aim primarily to show the robustness of the method to a wide variety of scripts and process of transforming our framework to new scripts.

1.1 Offline Recognition

Handwriting recognition is the ability of a computer to interpret handwritten text from sources such as paper documents, photographs, touch-screens and other devices. In offline recognition, text is recognized solely from digitally stored image data, usually from a scanner or a camera. Offline Handwriting Recognition (OHR) also contributes to task automation, such as postal address verification, bank check processing, document translation, digitization and manuscript archiving. Several industries are adopting this technology to batch process forms and applications. This also helps us preserve historical documents and disseminate the wisdom and saga of our ancients. Many machine printed historical documents are digitized and then treated with handwriting recognition techniques to make them more accessible. Overall this is a vividly promising technology, particularly in the fields of knowledge, education and research.

1.2 Bangla Writing System

Although the methodology of offline character spotting we propose is applicable to almost any script, the fundamental technique and tools were developed initially to work with the Bangla script, a connected script with diacritics and conjuncts. Therefore, here we present an overview about this writing system. Bangla, also called Bengali, is one of the most used languages in the world. With over 272 million people, it is the 7th most spoken language in the world [3]. The Bangla script, used also for the Assamese language, is the fifth most widely adopted writing system in the world [4]. It is the national and official language of the People's Republic of Bangladesh, and official language of several states in India such as West Bengal, Tripura, Assam and Andaman. Bangla belongs to the Abugida class of writing systems. It is written from left to right. The script is an alphabetic script that has 11 vowels, 10 vowel diacritics, 39 consonants, several hundred consonant conjuncts, more than 10 consonant diacritics, 10 numeric digits and several punctuation marks. There is no upper or lower case distinction of characters in the Bangla script.

Fig 1 shows the 11 vowels, 39 consonants and 10 numeric digits of the Bangla script. As an Abugida script the vowel graphemes are not used as independent letters as shown in **Fig 1** (a), rather as diacritics modifying the vowel inherent in the base consonant letters they are attached to. **Fig 2** shows an example of how this compares with the Latin script and how the consonant ' ' (Ma) appears with all possible vowel diacritics respectively. When two or more consonants are adjacent without any vowel between them, they form a compound consonant or consonant conjunct and usually the form of the character is modified. **Fig 3** (a) shows an example of how Bangla conjuncts work

compared with Latin. When two or more consonants are adjacent without any vowel between them, they form a compound consonant or consonant conjunct and usually the form of the character is modified as shown in **Fig 3** (b). There are a few consonants which have their own diacritics while forming a conjunct like the vowels. Some of them have even more than one form of ligature. The punctuation and other symbols are mostly similar to those used in the Latin script except the 'Period' symbol looks like '|'.

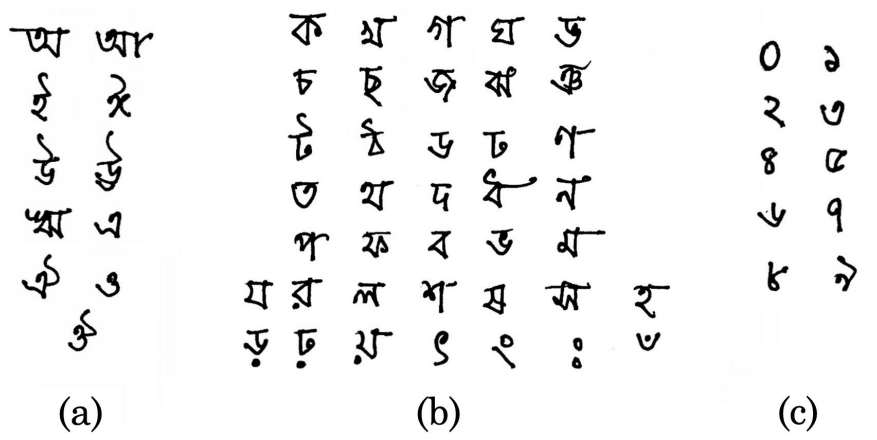


Fig. 1 Handwritten (a) 11 vowels, (b) 39 consonants and (c) 10 numeral digits in the Bangla script.

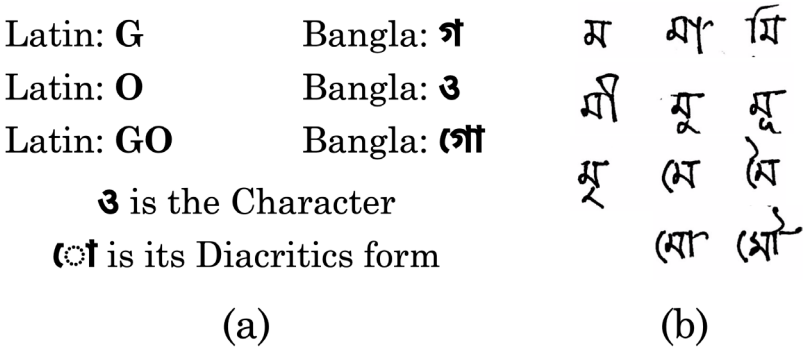


Fig. 2 (a) Bangla diacritic use compared with the Latin script. (b) Diacritical forms of vowels with the consonant 'Ma'. The 'ˆo' sound is inherent to the solo consonant in the top left. Other vowel sounds are formed by attaching other diacritics to it.

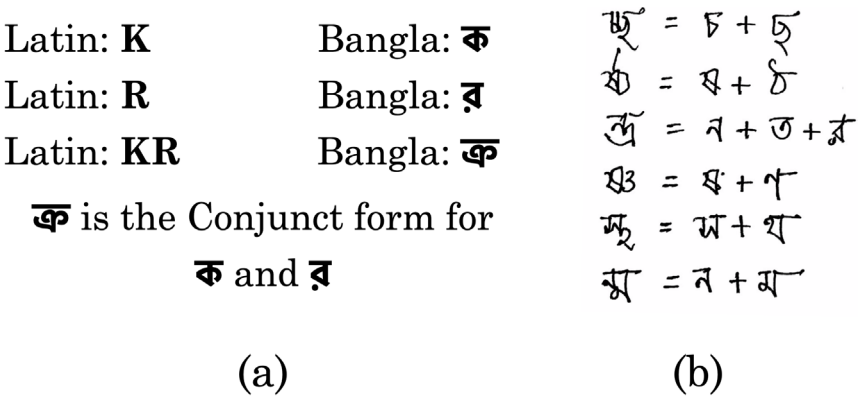


Fig. 3 (a) Bangla consonant conjuncts compared with the Latin script. (b) A few consonant conjuncts in the Bangla script along with their component solo constants.

1.3 Scripts with Similar Properties

Bangla is an eastern Indo-Aryan language spoken in many parts of the Indian subcontinent. Lots of other scripts in this region share similar attributes with the Bangla writing system. Some widely used scripts like Devanagari (for the Hindi language) and Gurmukhi (for the Punjabi language) share close resemblance with Bangla. Hindi and Punjabi are the fourth and tenth most spoken first languages in the world. Many other Indo-Aryan scripts like Gujarati, Kannada, Malayalam, Oriya, Tamil, Telugu, etc. also have similar structural properties with Bangla. All of these scripts have a common root of the Brahmi script (which is no longer used). A few handwriting samples of such similar scripts are shown in **Fig 4**. The attributes these other scripts share with Bangla makes it highly likely any recognition system developed for one of them should strongly influence development of, if not be exactly applicable, to others.

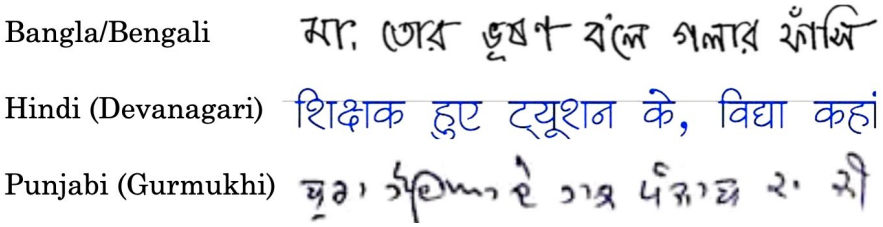


Fig. 4 Handwriting samples of Bangla, Devanagari and Gurmukhi scripts.

1.4 Difficulties with Recognition of Indic Scripts

Many Indic scripts like Bangla have a large number of unique symbols compared to, for example, the Latin script. This is primarily because of the vowel diacritics and consonant conjuncts. Diacritics can appear anywhere relative to the consonant and there are no definitive rules for predicting the consonants' shape or location from the conjuncts. A conjunct can also have its own vowel diacritic. There can be up-to 5 different components inside such a structure. It is almost impossible to separate these building block components from the combined structure, and these are better treated as unique symbols for machine learning. Therefore, the number of uniquely shaped glyphs needed to be included in machine learning for these scripts is often more than 2000. Tools like Deep Learning depend on having hundreds of labeled data samples for each class. Such a large dataset is impractical to form just for a single script. This is the primary reason why we have not seen any noticeable progress in offline recognition for scripts like Bangla.

Furthermore, most Indic scripts are connected scripts by nature. Unlike scripts like Latin, these cannot be written in a way where characters in a word do not touch each other. This makes any kind of segmentation-based approach very complicated. Along with that, many characters and symbols look very similar to each other. In addition to inter-writer variations, many of these scripts have several writing styles which completely change the appearance of many characters and conjuncts. All these together make offline recognition very difficult for Abugida and similar scripts, and most of these have seen little to no progress.

1.5 Korean/Hangul Writing System

In order to demonstrate how our proposed character spotting with autonomous tagging approach can easily be morphed to apply to different kinds of scripts, we chose Korean since it is another challenging script that has a very different structure than Bangla. This script is known as Hangul/Hangeul in South Korea and Chosŏn'gŭl in North Korea. This is the official script of Korea and a co-official writing system in the Yanbian Korean Autonomous Prefecture and Changbai Korean

Autonomous County in Jilin Province, China. Some parts of Indonesia and Taiwan also use this script. Korean is an alphabetic script with 28 basic characters or Jamos. These are arranged into syllables by combining up-to 6 characters. This syllabic arrangement happens in two dimensions, i.e. they can combine both horizontally and vertically, which is one of the most distinguishing features of Hangul. However, unlike Bangla, the shapes of the original characters do not change based on the character sequencing process. This particular property makes Korean an easier script to process with the proposed character spotting method than Bangla. Furthermore, Hangul characters are written separately and are not inherently connected like Bangla or cursive Latin, although like in most scripts when handwritten the characters touch.

The Korean alphabet and a handwriting sample are shown in **Fig 5**. The Hangul syllable is composed of three constituent parts: lead consonant, vowel, post-consonant. The top left is the first consonant. If the vowel is a vertical symbol it would appear on the upper right of the syllable, a horizontal vowel would appear below the lead consonant, and a compound vowel would appear in both geometric places. The post-consonant (if the syllable includes one) follows at the bottom. Consonant positions may contain single or paired consonants.

(a) Korean alphabet and composite symbols

(b) Korean handwriting sample

Fig. 5 (a) 24 basic characters and 16 compound characters in the Korean/Hangul alphabet. The green colored ones are used for K-Net training and the orange-colored ones are recognized by their vowel components. (b) sample handwriting.

2 Literature Review

2.1 Overview: It's an Unsolved Problem

Despite of lots of interest and activities, offline recognition is still considered to be an unsolved problem today. While popular scripts like Latin received significant attention and progress, most others like the Abugida scripts have seen close to none. For example, even though Bangla is one of the most widely used scripts in the world, the offline recognition progress is based primarily on scattered solutions which work only under specific terms and conditions. No one has ever reported a framework for Bangla offline handwriting that works in a broad range of contexts. The situation is similar for many other scripts. The following subsections discusses the fundamental approaches of offline recognition along with some of the most successful techniques and notable achievements for different scripts. We focus the literature review mostly on the Indic originated scripts and Korean as a background for our proposed approach. It should be noted that, in most cases, the works we present are not directly comparable with each other since they vary in number of target classes, nature of the script in concern, the amount and quality of data used and many other differential factors.

2.2 Segmentation-Based Approaches

Segmentation-based approaches usually go through a two stage process: first segmenting individual characters from a word image and then applying a classifier to recognize each isolated character. The

process of character segmentation can be very difficult to achieve cleanly. This is true for almost every script, especially ones in which the characters are connected like Bangla, cursive Latin and Arabic.

Although segmentation-based approaches are usually script dependent, many of the techniques used in Indic writing systems can be useful for others since they share many common core attributes. Often a key trait, a horizontal line that runs on top of the characters to connect them in a word (known as Matra, Shirorekha, etc.) is utilized to segment characters from a word. Malakar et al. used a fuzzy membership-based Matra segmentation followed by a rule-based lower zone separation to segment unconstrained handwritten Bangla text and obtained a F-score of 91.12% [5]. Mitra et al. exploited the fact that a successful identification and removal of the Matra separates most character components from a Bangla word [6]. Kohli and Kumar used a Segmentation Facilitate Feature (SFF) to find junction pixel in handwritten Devanagari text and obtained around 90% accuracy character segmentation accuracy [7]. Mahto et al. used horizontal and vertical projection features to segment Gurmukhi characters and obtained an accuracy of 91.4% on their word dataset [8]. Javia et al. proposed a Faster RCNN object detection model to segment handwritten Gujarati text [9]. Other Indic scripts like Malayalam, Tamil, Telugu are coherently isolated scripts, but still some attention is required to address the occasional overlaps while handwritten. In theory, these kind of frameworks can also be expanded to segment consonant conjuncts, but it is often avoided due to the level of complexity, and the compounds are treated as unique objects which is more convenient for most cases. There are only a few works (such as [10, 11]) which attempt to segment the fused consonants.

Another approach of doing OHR is N-gram modeling, which is used to aid recognition by segmenting words into N adjacent characters, with N usually being 2 (bigrams) or 3 (trigrams). This capitalizes on handwriting being statistically more similar when people write a set of characters than how they write individual characters, since the neighboring characters influence the writing process. This approach is very useful for the scripts with fewer characters like the Latin script, where with 52 character symbols there can be 52×52 bigram patterns, of which many are seldom or never used in common text. Things are much more difficult for scripts like Bangla where there are more than a thousand different symbols (due to the consonant conjuncts and diacritics), and the conjuncts are already bigrams, trigrams or quadrigrams [12]. This makes the model synthesizing process extremely complicated and impractical.

In theory, character segmentation with an isolated character recognition process can be merged into a full transcription unit. Usually different Convolutional Neural Network (CNN) architectures are most widely used for classifying isolated characters. For Bangla, Ghosh et al. achieved a 96.46% accuracy in recognizing 231 character classes (171 compound, 50 basic and 10 numerals) from CMATERdb [13] using a MobileNet architecture [14]. Mishra et al. reported a 99.72% accuracy on the Devanagari Handwritten Character Dataset with 46 different character classes using a ResNet architecture [15]. Mahto et al. presented a CNN architecture and obtained 98.5% classification accuracy with 35 character classes with the Gurmukhi script [16]. Rani et al. achieved about 90% accuracy with 188 Kannada character classes using a VGG19 network [17]. Similar works can be found for other Indic scripts, like Tamil [18], Telugu [19], Malayalam [20], etc. Chauhan et al. proposed a CNN architecture, called HCR-Net which works on multiple scripts like Bangla, Punjabi, Hindi, Telugu, Marathi, etc. [21]. A common pattern we observed from this survey is the recognition accuracy goes lower as more classes are added, especially when the conjunct characters are also included. All these approaches can be used for unconstrained handwriting recognition if followed by a character segmentation process although the overall performance will be dependent on the losses of each stages.

Like many Indic scripts, Korean is also an inherently isolated writing system; the syllables and the characters do not touch each other (except accidentally). Therefore offline recognition can be achieved just with an isolated character recognizer. Park et al. compared fifteen character normalizations, five feature extraction and four classification methods and evaluated their performance on two public handwritten Hangul datasets, SERI and PE92. They obtained 93.71% and 85.99% syllable recognition peak accuracy respectively [22]. Kim and Xie used deep learning based modeling on the SERI95a and the PE92 datasets and achieved 95.96% and 92.92% recognition accuracy respectively [23]. The highest reported accuracy (97.76% on SERI and 97.90% on PE92) was presented by Dziubliuk et al. using a multidimensional sequence learning architecture [24]. In fact, isolated symbol recognition using deep learning for any script is now considered to be a solved problem if enough data is available. This is because of the "near-human performance" demonstrated with the MNIST dataset (handwritten Latin digits) by many researchers in the last decade, although this has only 10 classes.

2.3 Segmentation-Free Approaches

Segmentation-free approaches attempt to recognize a whole unit (usually a word) rather than going for segmenting into smaller chunks. For Indic scripts, the most common trend is to train on whole word images of some frequently used words, hence the process has a vocabulary restriction. Pramanik and Bag experimented with several different neural networks and achieved a 98.86% accuracy with a ResNet50 architecture for the task of Bangla city names recognition [25]. Sharma et al. used a CNN with ADAM and Stochastic Gradient Descent (SGD) optimizers for recognizing Gujarati city names [26]. Similar works are presented for other Indic scripts like Devanagari [27], Malayalam [28], Marathi [29], etc. Usually the performance of whole word recognition decreases with the increasing number of words to identify. Therefore, a generic word recognition without vocabulary restriction using these approaches is impractical. One such attempt was made by Adak et al. They obtained a WRA close to 90% with a closed set of words, but the performance dropped to around 70% when they tried to apply the same framework for unconstrained Bangla [30].

In general, the Long Short-Term Memory (LSTM) or HMM are considered to be the one of best tools for unconstrained offline recognition. However, most successful works using such architectures can be found for purely alphabetic scripts like Latin, French or Arabic. For complicated scripts like Bangla, these architectures are still restricted to problems like recognition of whole word, online handwriting, offline but machine printed text, or isolated characters. One reason is because the LSTM or HMM networks depend on character modeling and finding the characters from a word image. This often generates a stream of characters which includes many false detections and duplicates of similar characters which are usually decoded with a grammar or language-based model. Most of the Indic scripts do not have a strong model ready to be used. Another big problem comes from the fact that an LSTM network sweeps the space horizontally, which can not tackle the different character-character combinations which come frequently for scripts like Bangla or Korean. When characters or other symbols share vertical spaces, the whole combination has to be treated as one character if it is to be modeled with an LSTM or HMM-based architecture. This again leads to the problem of having a very big number of classes to be modeled and thereby the dataset requirements grow while the recognition speed and performance falls. Therefore, although theoretically sound, no notable offline transcription works have yet been reported for these complex scripts.

Another recently introduced segmentation-free approach is based on character spotting, which uses an object detection network to locate and identify each character element in a word image, which is assembled into a transcription. When we introduced this approach in [1], we obtained a CRA of 91.1% on Bangla. Details of this idea can be found in Section 3. Mondal et al. applied this approach to

the Latin script using a YOLOv3 object detection network obtaining over 90% CRA [31]. This approach is not restricted by vocabulary; therefore it offers a truly unconstrained form of recognition. The problem is, the training for character spotting depends on character level location metadata, which makes the dataset preparation process extremely cumbersome. In this paper, we attempt to fix this problem by adding an autonomous tagging layer on top of the character spotting approach and make the process of developing an unconstrained offline handwriting recognizer very convenient for any alphabetic script. We demonstrate this and compare with our former results in [1], and show how it can easily be applied to other alphabetic scripts using Korean as an example.

3 Offline Recognition using Character Spotting

As we discussed in Section 2, offline handwriting recognition is still considered to be an unsolved problem. There are partial solutions and scattered works for many scripts, but no unified method has ever been demonstrated which can work on any alphabetic script. Our primary ambition while designing an offline recognizer is to address this issue. We introduced an approach which we call "Character Spotting" in [1] which uses an object detection network to spot different character elements in a word image. Here we expand our original research to build a complete segmentation-free offline recognizer which can work for potentially any alphabetic script. This is a very easy and intuitive approach that neither depends on prototype words, nor is it restricted to a limited vocabulary. Back in [1], we demonstrated this idea on a portion of the Bangla script which included all the basic characters and only a few high frequency conjuncts. Here, not only we do expand the framework to include practically the entire Bangla script, we also transform and test this approach with the Korean Hangul script, which has a drastically different structure than Bangla. One of the biggest bottlenecks of our initial approach was labeling the dataset, because the network was dependent on accurate character locations which requires a massive amount of manual labor and time to prepare. Here we introduce a complete autonomous system that removes the need for manual data tagging. We call this "Autonomous Tagging" that makes the development process using our framework very easy and approachable for any alphabetic script. The following subsections present the details of all the steps, showing how we achieved a complete unconstrained offline handwriting recognition system for Bangla, how we transformed this system to recognize the Hangul script and how this framework can be easily adapted for any alphabetic script using autonomous tagging and character spotting.

3.1 The Basic Idea of Character Spotting

The basic idea of our character spotting offline recognition method is first to identify and locate individual character elements in a word image and then to use the location information to combine the characters to produce a transcription [1]. This concept is explained in **Fig 6** with the Latin script. For the example word "Good", the object detection network attempts to find the letters from "A/a" through "Z/z" in the word. This process is not sequential; the character spotting algorithm runs for all possible characters simultaneously. After it finds all the matches, we use the classes of the characters found (one "G", two "o"s and one "d" in this example) and their relative location information (like the "G" is detected to the left of an "o") to obtain a transcription of the word.

This simple idea is strong enough to be implemented for any alphabetic script with few adjustments as we will demonstrate in the following subsections. Character spotting is, in a sense, analogous to word spotting algorithms, but is much more powerful since it is not restricted by word prototypes and thus potentially covers an entire script.

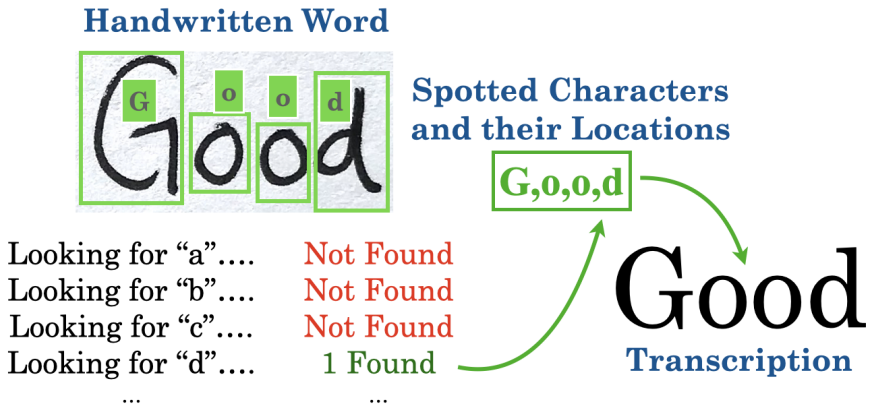


Fig. 6 Basic concept of the presented offline character spotting recognition method. An object detection network attempts to find character matches in the word and the transcription is formed with the detected character classes and their relative location information.

3.1.1 Implementation for the Bangla Script

As discussed in Section 1.4, because of the diacritics and conjuncts the Bangla script can effectively have more than 2000 different classes on which the object detection network needs to be trained. Almost all Indic scripts have this attribute. This is a problem since for training we require hundreds of samples for each possible class. To address this issue, we prepared two networks, one for the characters and one for the smaller symbols (like diacritics) which appear around the other characters keeping the base shape unchanged [1]. We call these Character Network (C-Net) and the Diacritic Network (D-Net) respectively. The lists of symbols that each of these networks are trained to detect are shown in **Table 1**. On each word image, both of these object detection networks are applied sequentially. The C-Net first locates and identifies the characters from the list, and afterwards D-Net does the same with the diacritics. Therefore, even if we trained C-Net and D-Net from the same compound characters (a basic/conjunct character with a diacritic) from repeatedly showing the same part in different context, the C-Net only spots the characters and D-Net only the diacritics when tested on unseen words. This process is schematized in **Fig 7**.

Once we obtain the detected classes from C-Net and D-Net along with their location information, a transcription is formed as shown in **Fig 8**. This two-network-approach breaks the 2000 class problem into two smaller chunks, analogous to 20 × 100, and thereby the dataset requirement becomes much more manageable. This is applicable to any Abugida script, since they all share similar attributes. Another strong point of this approach is the individual networks are trained not only to spot the target characters, but also to ignore the surrounding characters/diacritics. This pattern of training makes the character spotting approach robust and insensitive to ground truth position tagging accuracy (demonstrated in Section 4.1) and allows an autonomous method of tagging to be implemented to avoid intensive manual labor during dataset preparation (demonstrated in Section 4.2).

In our previous work in [1], we included 19 high frequency conjuncts from the Boise State essay script dataset in training C-Net. The Bangla script has several hundred conjunct classes. In order to make a more comprehensive offline recognition framework, we analyzed a large volume of written documents and defined a character set which covers over 99% of the entire Bangla script. These were added to the Boise State dataset through a set of conjunct specific words. Details are presented in Section 5.1.2. This approach reduced the class size by roughly 30% and made the whole process of dataset preparation as well as training much more convenient without sacrificing coverage of the Bangla script. The conjunct column in **Table 1** presents the reduced conjunct list. The black colored

ones were previously used in [1] and the blue colored ones are added and used with the original dataset for our current recognition framework.

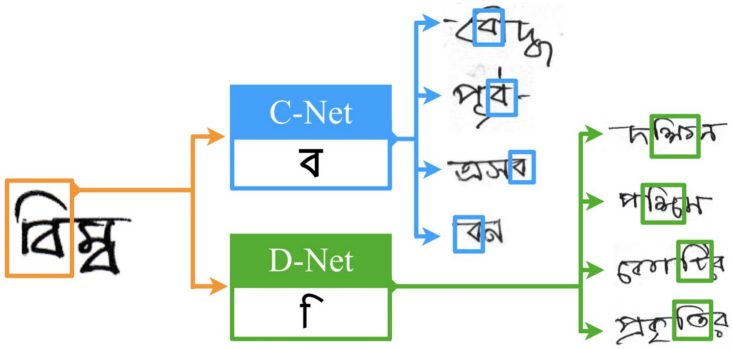


Fig. 7 C-Net and D-Net work on the same image, but are trained to detect different symbol classes [1].

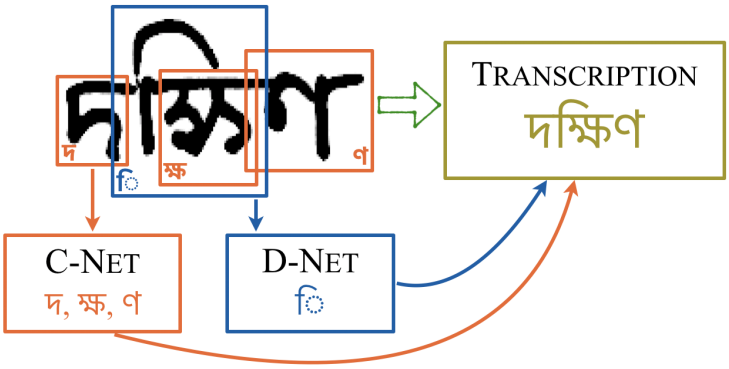


Fig. 8 Detected symbols from C-Net and D-Net are merged to form a transcription [1].

Table 1 Class distribution for C-Net and D-Net training. The black colored characters are from the essay scripts from the Boise State dataset used in [1]. The blue colored ones are from the newly added conjunct word documents of this dataset.

C-Net Classes			D-Net Classes		
Basic	Conjunct	Punctuation	Basic	Diacritic	Conjunct
অ, আ, ই, ঈ, উ, ঊ, এ, ঐ, ও, ঔ ক, খ, গ, ঘ, ঙ, চ, ছ, জ, ঝ, ট, ঠ, ড, ঢ, ণ, ত, থ, দ, ধ, ন, প, ফ, ব, ভ, ম, য, র, ল, শ, ষ, স, হ, ড়, ঢ়, য়, ৎ, ং, ঃ	ন্দ, শ্চ, ঙ্গ, ধ্ব, গ্ধ, ষ্ট, ষ্ঠ, ষ্ম, প্ল, ত্ত, ন্ন, দ্ব, ব্জ, ক্ষ, ত্ত, ট্ট, ষ্ণ, দ্ধ, ন্ড ক্ক, ক্ট, ক্ত, ক্ব, ক্ত, ক্ল, গ্ন, গ্ধ, গ্ন, গ্ল, ঘ্ন, ঙ্খ, ঙ্ঘ, চ্চ, চ্ছ, জ্জ, জ্ঞ, জ্ব, ঞ্চ, ঞ্ছ, ঞ্জ, ড্ড, ণ্ট, ণ্ঠ, ণ্ড, ণ্ন, ণ্ম, ণ্ব, ত্ত, ত্ন, ত্ম, ত্র, থ্ব, দ্দ, দ্ভ, দ্ম, ন্ট, ন্ঠ, ন্ত, ন্ধ, ন্ব, ন্ম, ন্দ, প্ট, প্ত, প্ন, প্প, প্স, ষ্ঠ, ফ্ল, ব্জ, ব্দ, ব্ধ, ব্ব, ব্ল, ম্ন, ম্ভ, ম্প, ম্ফ, ম্ব, ম্ম, ল্ল, ল্ক, ল্প, ল্ট, ল্ড, ল্ব, ল্ল, ল্ফ, ল্ব, ল্ভ, ল্ম, শ্ছ, শ্ন, শ্ম, শ্ল, ষ্ক, ষ্প, ষ্ফ, ষ্ঠ, স্ক, স্ব, স্ত, স্থ, স্ন, স্প, স্ফ, স্ম, স্ল, স্থ, হ্ণ, হ্ন, হ্ম, হ্ল, হ্ম, হ্ম, চ্ছ, জ্ঞ, ত্ত, দ্ভ, দ্ম, ন্ত, ন্ধ, ন্ব, ষ্ক, স্ক, স্ত্র, স্প্ল	‘, (কমা)’, ‘। (দাড়ি)’, ‘- (হাইফেন)’	ঁ (চন্দ্রবিন্দু)	া, ি, ী, ু, ূ, ে, ৈ, ো, ৌ, ৃ	রেফ (e.g. র্প), র-ফলা (e.g. প্র, exception ক্র and ত্র), য-ফলা (e.g.প্য), য-ফলা আ-কার (e.g. প্যা) *(প is used for just an example base character here)*

3.1.2 Implementation for the Korean Script

To develop a Korean OHR system, we used the same framework as for Bangla, but with different strategies based on the unique attributes of the Korean script. As we described in Section 1.5, the Korean script does not have any diacritics, but has syllables where the 28 Jamos letters are arranged in a two-dimensional structure. This results in more than 2000 different possible combinational arrangements, like Bangla, and therefore faces the same the problems. The Jamos are written separately and are not inherently connected like Bangla as can be seen from **Fig 5**(b). Also, the Jamos do not change their shapes based on position or neighbors like many Bangla characters. Therefore, rather than trying to recognize a Korean syllable as a whole, we tuned our object detection network to locate and identify the Jamos inside the syllable. Since the number of Jamos are much fewer than

the number of possible syllabic combinations, our offline recognition process can be achieved using a much smaller dataset and shorter training than the traditional approaches. Therefore, while the basic idea remains the same as with the Bangla script, the execution was adapted to better treat the attributes of the Korean script.

Because there are no diacritics in Hangul, we prepared a single network we call the Korean Network (KNet), to recognize 33 Korean character classes. These are the green characters from **Fig 5**(a), which includes the 14 consonants, 10 vowels, 5 tense consonants and the first 4 complex vowels. The last 7 complex vowels (shown in orange) are recognized by their vowel components. The rest of the architecture and tools are identical as for Bangla. The recognition process is the same regardless of how many Jamos are in the target syllable. **Fig 9** shows an example syllable which is made of 4 Jamos. Rather than trying to recognize the syllable as a whole, the K-Net spots those 4 Jamos, then from those detections we re-construct the syllable. In this way, our object detection network only has to master spotting those 33 classes, rather than being trained to recognize more than 2000 possible syllable classes, and the segmentation process is avoided.

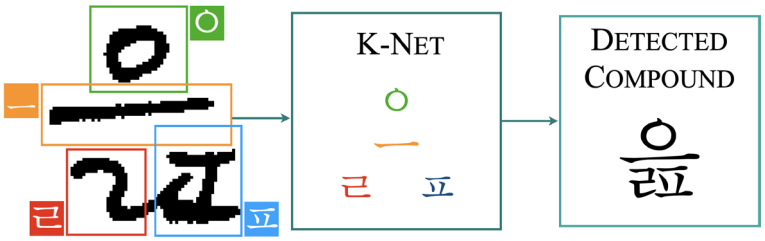


Fig. 9 The Korean offline recognition process. This example shows a syllable made of 4 Jamos. Instead of recognizing the whole syllable, the K-Net only spots the Jamos. The compound syllable is constructed from the detected classes and their corresponding locations.

3.1.3 Implementation for any Alphabetic Script

This idea of character spotting is very flexible to the nature of the script. All the Abugida scripts (with vowel diacritics) or scripts with similar attributes can share the same framework used for Bangla by spotting characters and diacritics with two detection networks. Pure alphabetic scripts like Latin, Arabic and Korean can be implemented using just a single detection network trained on their alphabets. Writing from right to left (like Arabic, Syriac), top to bottom (Kulitan, Nushu) or 2D Korean does not change the design core, just the order the spotted characters are compiled. Therefore any alphabetic or alphasyllabary script (which uses a limited list of symbols) can potentially be implemented using this proposed approach with few adjustments.

3.2 Underlying Framework

The hierarchical architecture of this offline handwriting recognition framework is presented in **Fig 10**. The core of this framework is an object detection network which is trained to detect the characters or symbols. Afterwards, the detection results are compiled to form a transcription which requires script specific implementation. At the end, one can include optional post processing, like a spell checker, to further improve the result.

The object detection network is developed from a pre-trained neural architecture using transfer learning. Then it is trained using an annotated dataset. The dataset annotation can be obtained from a collection of handwritten documents for a script subjected to our proposed autonomous tagging process. The green boxes in **Fig 10** signify the portions of this framework which are script independent, and the blue boxes highlight where some script specific treatments are needed.

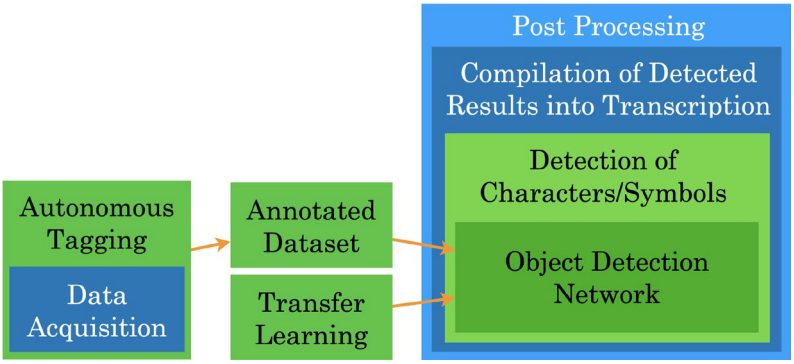


Fig. 10 Hierarchical building block of the offline Handwriting Recognition Framework. Green indicates the sections which are script-independent and blue indicates where some script specific attention is required.

3.2.1 The Object Detection Network

We used the Faster RCNN object detection network we presented in [1]. Although it is computationally expensive to train, applications like handwriting recognition often benefit more from faster recognition than a quickly trained network. Using Faster R-CNN in our framework is a choice, not a necessity. Any other similar object detection networks (e.g. Mondal et al. used YOLOv3 for Latin [31]) can also be used for with our approach to obtain similar performance.

Deep Learning is notorious for being slow to train. This issue can be mitigated by transfer learning, where the neural weights of an already trained network are retrained for another problem. Not only does it make the process of network training much faster, in most cases it ends up producing a better design as a whole. In our designs, we used VGG-16 [32] pre-trained with the ImageNet dataset [33]. VGG-16 is a widely used neural network. It is sufficiently large to handle the large number of classes in Bangla, and not so large that it would be considered an overkill. The choice of this particular network is not crucial for our design and potentially many other pre-trained networks like VGG-32 or RESNET could have been used to obtain similar results.

For our requirements, the series VGG-16 architecture was remodeled into a DAG (Direct Acyclic Graph) structure to implement a Faster R-CNN network, shown in **Fig 11**. The bounding boxes around potential objects in an image are handled with the Region Proposal Network (RPN) within the Faster R-CNN. A region proposal layer has two inputs – the classification scores produced by the RPN classification branch and the bounding box deltas produced by the RPN regression branch. A RoI (Region of Interest) max pooling network is used to output fixed size feature maps for all rectangular ROI within the input feature map in a Faster R-CNN architecture. In our design, the features extracted from the ReLU5 3 (Rectified Linear Unit) layer was processed by a RoI pooling layer with 7 × 7 feature map output size replacing the last max pooling layer from the original VGG-16 architecture. This is the same approach we presented in [1] and all the networks (C-Net, D-Net and K-Net) were formed with this architecture.

C-Net, D-Net and K-Net were all trained identically except for the difference in number of classes. We used the Stochastic Gradient Descent with Momentum (SGDM) for optimization with a momentum of 0.9. The learning rate was set to 0.001 and we ran all the experiments for 10 epochs. During this process, a negative training was defined with an overlap range with Intersection over Union (IoU) which is:

$$IoU = \frac{Area\ of\ Overlap}{Area\ of\ Union} or, \frac{area(A \cap B)}{area(A \cup B)} \quad (1)$$

where A and B are bounding boxes of the region proposal and values from the ground truth file. Cases with an overlap ratio of 0.6 or lower were used for negative training. We used 64 region proposals to randomly sample from each training image. All these training parameters are kept identical for all the experiments as it was in our previous work in [1].

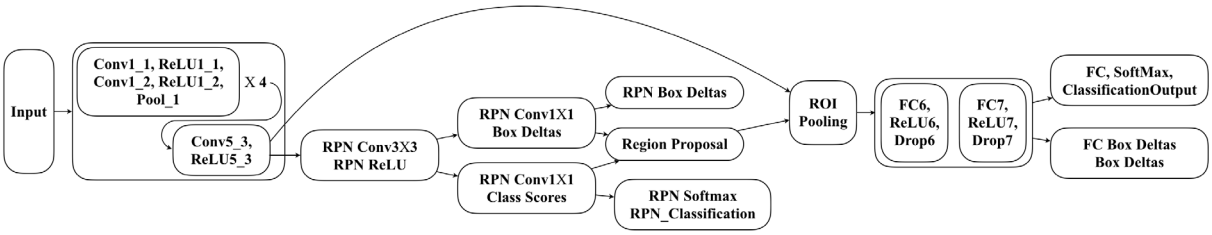


Fig. 11 Layer graph of C-Net, D-Net and K-Net, transformation of VGG-16 to a Faster R-CNN [1]

3.2.2 Compilation of Detected Results into a Transcription

As the networks (C-Net, D-Net or K-Net) detect the classes and locations of the characters and diacritics present in a word, the information is compiled into a transcription or plain text representation of the word image. To achieve this, we stepped through a series of processing steps similar to what we did in [1]. An example schematic of this compilation process is shown in **Fig 12**.

All detections are returned with a confidence value. Detections with confidence below 70% for all the networks are considered unreliable and are discarded in the compilation phase. Many of the returned detections spatially overlap with each other. Some overlaps were expected such as a C-Net and D-Net overlap when there is a character/conjunct with a diacritic. Overlaps from a single network indicate multiple characters in one place. This phenomenon is expected for both Bangla and Korean (and most other scripts), because many characters/elements visually look like extensions of other characters/elements, and Bangla conjuncts often look like individual characters merged together. An analogous example is the visual relation between Latin ‘c’, ‘e’ and ‘o’. A sample of C-Net detection overlaps is demonstrated in **Fig 13**. We keep the largest bounding box, discarding the smaller detected bounding boxes that it encapsulates or overlaps with regardless of their confidence score. The overlap is computed using Eq. (1).

At this stage we implemented some script specific treatments for Bangla in order to put the transcription in proper order and mitigate some obvious detection errors. The default ordering for Bangla is obtained from sorting the xmin values of the detected bounding boxes. For the following cases, we utilized some properties of the script and altered this default scheme:

1. With Unicode, diacritics are always encoded after their base characters. Visually diacritics can appear before or after and in many different orientation (like above, below, around, etc.). With this observation, whenever detection from C-Net and D-Net overlapped (Eq. (1)), the C-Net results were ordered first. One exception to this rule is for the class ‘ ’ (Bangla conjunct ‘ref’). This is actually a conjunct that behaves more like a diacritic (keeps the base character unchanged). Therefore we processed this class with the D-Net. With Unicode the character associated with ‘ ’ actually appears before the associated character. Hence, for this character the D-Net results are placed prior to the C-Net ones.
2. Vowels cannot have a diacritic, therefore D-Net results that overlap with a vowel are removed.
3. There can never be two consecutive diacritics in Bangla. If the compiled detection appears in that way, either there is a missing consonant between them or at least one of them is a false detection. For overlapped or tightly spaced detection, the diacritic with lower confidence is removed.

All these accommodations described above are different than a spell checker. A spell checker reduces the scope of transcription to a fixed vocabulary. These are mostly script-specific attributes and still allow an open vocabulary. Therefore, any non-dictionary words (e.g. names, unorthodox spellings) will not be affected with our approach. Most Abugida scripts share similar properties; therefore similar rules can be implemented for other scripts too.

This step was not done for Korean, since we used Korean only to demonstrate the script-flexible nature of our character spotting recognition framework. The recognition performance of Korean or any other script can also be improved using a similar sequence of processing.

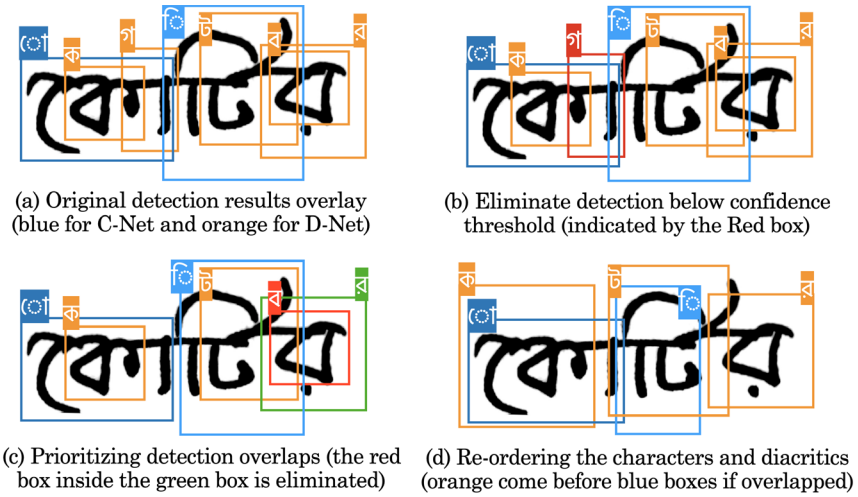


Fig. 12 An example case of post processing compilation steps for Bangla: (a) Original detections, (b) Eliminating detections below threshold, (c) Prioritizing detection overlaps, (d) Re-ordering the characters and diacritics.

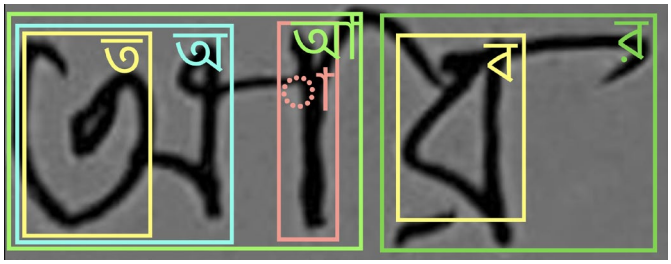


Fig. 13 Sample of the detection overlap issue. Green boxes are the proper detection and the all other colored boxes are detected look-alike sub-characters.

4 Autonomous Tagging

The primary weakness of our original character spotting approach [1] comes from its dependency on character level tagging of the dataset. We provided coordinates of a tight fit rectangular bounding box around each character. This was required for the character spotting to work, but to process data with such a detailed level of annotation requires an immense amount of manual effort and time. Furthermore, this process is also very error prone, and once an error happens its almost impossible to find and fix that. This sort of problem is quite common, and in many cases data processing is considered to be the biggest bottleneck of machine learning. Here we solve this problem for our framework by introducing a process we call “Autonomous Tagging”. This automates the process of drawing bounding boxes around each character on a handwritten word image. This is not a character segmentation method, and cannot be used with an isolated character recognizer. Rather autonomous tagging estimates the approximate location of a character in a word, and this only works because our character spotting algorithm is not too sensitive about the accuracy of the location of a character. Like

character spotting, the autonomous tagging process is also flexible and can be used for any alphabetic writing system. Here we demonstrated two approaches on the Bangla and the Korean scripts. Before proceeding with describing this framework, we study the detection performance of our networks with various level of imprecise bounding box annotations.

4.1 Sensitivity to Tag Precision

Most datasets with ground truth tags are concerned with getting the tightest and most accurate bounding boxes for each script element and connecting a label to it. As a first study we change the boundaries of the precise manual ground truth tags or bounding boxes to observe how critical the precision is to the recognition method. Each character from the Boise State dataset was labeled using a rectangular box. As shown in **Fig 14**, we increased (or decreased) the width of these bounding boxes by an amount of -10%, 10%, 20%, 30% and 40% (equally from both side) while keeping the height at the initial level (which is generally the word height).

The impact of changing bounding box widths on the recognition performance is presented in **Table 2**. **Fig 15** shows a plot of all these performance parameters versus the tag variation. The system does not work well when the boundaries are shrunk, since in many cases the defining attributes of the component get cut off from the edges. When box size is extended, the performance degradation is relatively small. Our objective with this experiment was to have a qualitative idea about the impact on detection performance if the characters are not tightly tagged and presented with fragments of the adjacent characters. We used this observation as a foundational idea in our autonomous tagging approach.

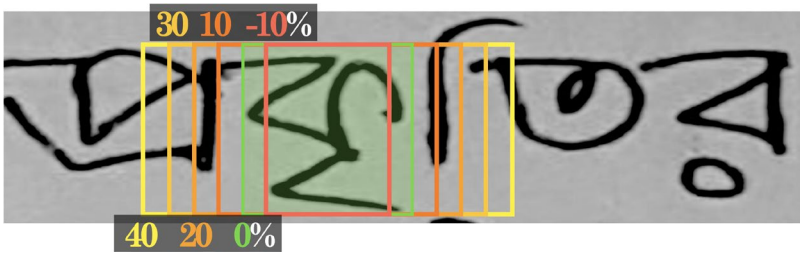


Fig. 14 Bounding box widths were varied from the green box indicating the accurate location to -10%, +10%, +20% and +30% as shown by boxes of oranges and yellows.

Table 2 Detection Performance with Tag Width Variation

Change in Width	C-Net		D-Net		Transcription	
	mAP	F1	mAP	F1	CRA	WRA
-10%	85.06	91.2	88.69	92.41	91.9	80.94
Accurate (0%)	91.41	95.08	92.77	95.38	93.61	86.8
10%	91.13	95.02	92.32	94.59	93.24	86.14
20%	90.04	94.35	92.06	94.38	93.07	85.9
30%	89.39	93.85	90.73	93.73	92.71	84.72
40%	87.44	90.08	89.04	90.42	92.39	82.28

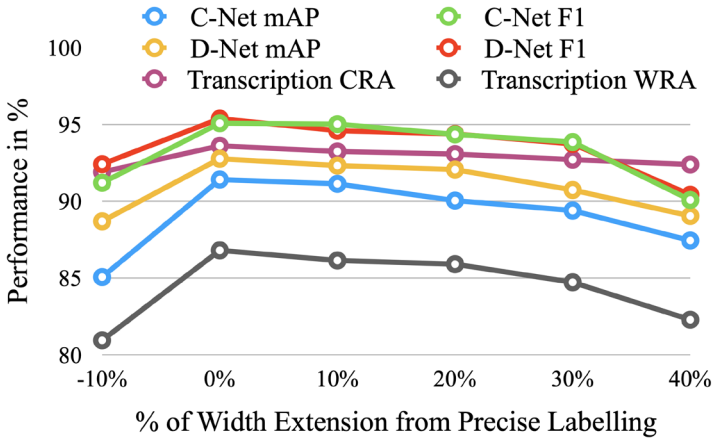


Fig. 15 Plots of detection performance with tag precision.

4.2 Autonomous Tagging Process

The autonomous tagging process first estimates the locations of the characters inside a word and then extends the boundaries, so that even allowing for variabilities in handwriting, the target character is most likely to be somewhere inside that extended bounding box. This is possible since we already observed that extending bounding boxes of ground truth tags does not considerably impact the detection performance.

4.2.1 Implementation for Bangla

The initial location estimates for the characters and diacritics for handwritten Bangla words are approximated from the relative character widths in a machine printed version of the ground truth text. The process is explained with an example in **Fig 16** for a three-character word. Here, W indicates the total width of the word composed by the three characters (including diacritics) with widths of X, Y and Z. The subscripts H, P and E represent the handwritten, printed and estimated character widths respectively. The widths of the printed characters (including diacritics) are measured from the machine generated font, from which they are proportionally imposed on the handwritten word with a width extension factor of η. All estimated widths are extended by η/2% on both sides except for the first and last characters of a word, which are extended by η/2% only on the interior edge. For the printed font we used 'Akaash' which we empirically found is one of the many fonts which has a nice match with typical handwritten shape proportions. Additionally most of the time people include a larger space before punctuation in handwriting than appears in machine print, therefore, we inserted one blank space before each punctuation mark to obtain a better estimate. For Bangla, we used η values of 20%, 30% and 40% to obtain 3 estimated location for each character and used all of them to train the CNet and D-Net. These η values are chosen reflecting our results in the tag sensitivity experiment described in Section 4.1.

Fig 17 shows a schematic workflow of how autonomous tagging works with the character spotting framework. This example shows the estimated boundaries around the Bangla character ' '. The object detection network is trained to locate the same character with a given boundary, but the boundary actually contains most of the target character and some extra parts around it. These parts can be a diacritic, or can be strokes from the prior or next character or diacritic. Furthermore, these extra undesired elements can be anywhere (left, right, top or bottom) and will not have a fixed pattern. This phenomenon over the iterations of neural training prepares the network to treat anything different from the target character as arbitrary and not important for the decision. A cost of training an object detection network in this format is the fact that the network does not learn to precisely locate the character. Therefore, the network can confidently predict the class, but only vaguely

predict an area within which the character is located as shown in **Fig 18**. However, this is not a problem for handwriting recognition, since we only need the relative positions of character elements with respect to the others.

The difference in performance from autonomous tagging versus precise labelling actually comes from the eccentric property of human handwriting. No matter how good the initial estimate is or how robust the network performs with width tampering, there will be some cases where the autonomous tagging misses the target character/diacritic completely or a major structural part of it. A similar situation can also happen from manual labeling since that process is very error-prone. In both cases, this issue can be solved by increasing the volume of the dataset. With more data, the ratio of proper labeling to mislabels gets higher and the network gets enough good samples to be effectively trained. Preparing larger quantities of data is significantly more convenient with an autonomous tagging framework than using manual annotation.

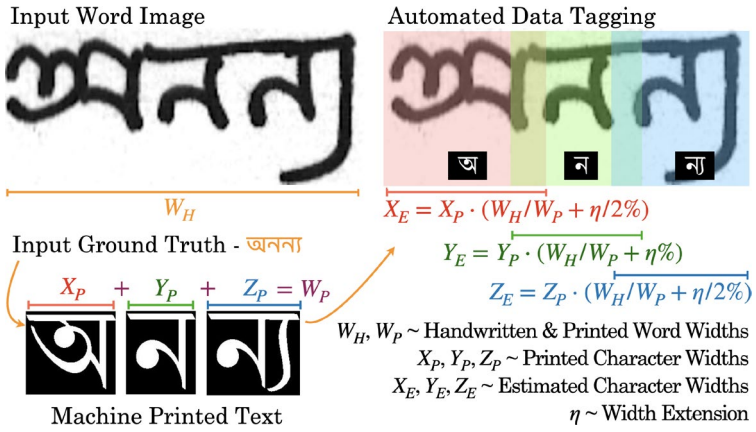


Fig. 16 Example of autonomous tagging for a three character Bangla word. Based on the character widths obtained from the machine printed text ($*_P$), the widths of the characters and associated diacritics in the handwriting are estimated ($*_E$).

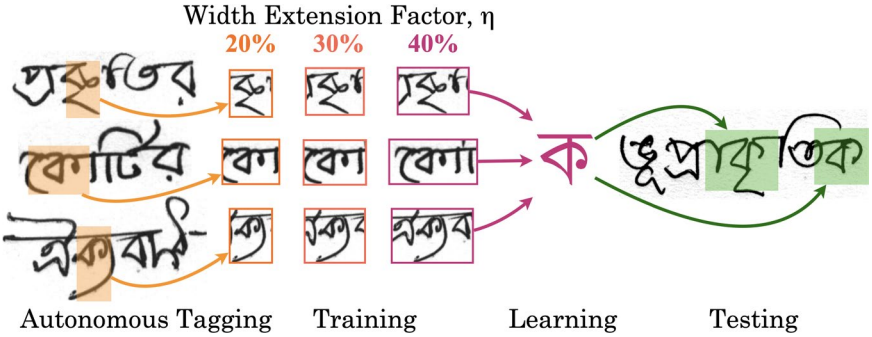


Fig. 17 Illustration of how autonomous tagging works. Each character is boxed with variable widths for training. The position of the learned character is shown in the test words.

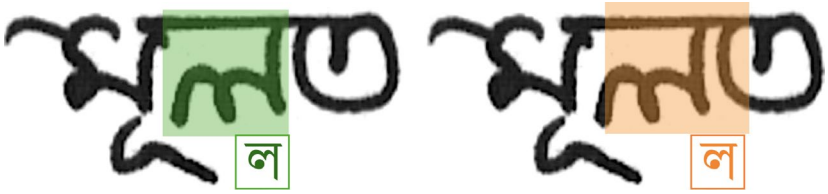


Fig. 18 Detection from the networks trained with manual (left) vs. autonomous tagging (right).

4.2.2 Implementation for Korean

For Korean we used a much simpler approach for the initial character size and position estimation. The basic characters in a Korean syllable usually appear in a well-defined grid of a limited number of shapes. We obtained a list from the Unicode foundation of how every Unicode Hangul syllable was composed of its parts. The height (or width) was then divided into 1, 2 or 3 zones and that distance apportioned to those symbols with some additional buffer size. This resulted in the 8 different grid structures of Hangul syllables shown in **Fig 19**. If there are two Jamos over the vertical (horizontal) span of the syllable, like in **Fig 19** (a) through (e), the initial zone height (width) estimate for each Jamo is set to 50% of the whole syllable height (width). This base estimate of 50% is used, even if the specific Jamos do not have the same height (width). For cases where there are three Jamos, like in **Fig 19** (f), (g) and (h), the initial height estimate is set to 33%. In other words, the initial estimated location of a Jamo is obtained by equally dividing the space (horizontally and/or vertically) based on the original structure of the compound syllable as shown by the grid shapes under each compound in **Fig 19**.

We increased the initial estimates to ensure the target Jamo is inside the labeled bounding box, just like we did for the autonomous tagging of Bangla characters. We only increased the estimated widths for the Bangla characters, where with Korean we increased both the estimated widths and heights because of its 2D writing style. The initial size estimate of 50% was increased to 60%, 70% and 75%, and the initial size estimate of 33% was increased to 40%, 45% and 50%. Therefore, we generated three different estimates for every Korean compound syllable image and all of these were used for training the K-Net. Machine printed fonts could have been used for Korean the same way we did for Bangla to account for when the Jamos do not have equal heights or widths, but this approach is simpler and works well.

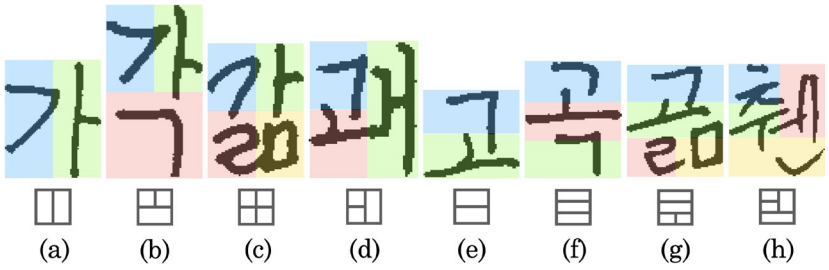


Fig. 19 Initial estimated bounding boxes of the Jamos from a compound Korean syllable. Widths and heights are divided into 2 or 3 zones based on to which geometric structure from (a) to (h) it belongs. Data samples are obtained from the Handwritten Korean/Hangul PE92 dataset [34]

5 Datasets used for the Experiments

For our first experiments with character spotting in [1], we used the essay scripts from the Boise State Bangla Handwriting dataset [2]. Since then, we expanded this dataset in different ways in order to achieve a more comprehensive training for Bangla. Here along with this extended dataset, we also used three other Bangla datasets to test our network more thoroughly. For Korean, we used one of the most famous datasets, PE92 [34], for both training and testing. Details of these datasets are described in the following subsections.

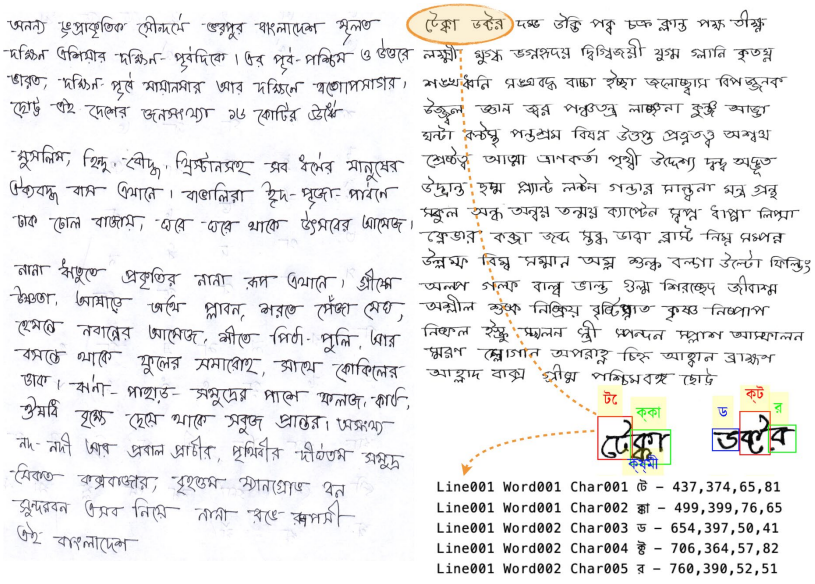
5.1 The Boise State Bangla Handwriting dataset

The Boise State Bangla Handwriting dataset was first introduced by us in [2]. Our primary motivation for this dataset was to make it comprehensive, versatile and open-to-all to provide a resource for the whole Bangla offline recognition community. The dataset can be accessed at

https://doi.org/10.18122/saipl/1/boisestate. Currently, this dataset contains three types of content: isolated character, essays and conjunct words. We used the later two for this work.

5.1.1 Essay script documents

This part of the dataset contains handwriting samples of an essay, which is carefully scripted to contain all Bangla basic characters (except ‘ ’, which rarely appears in its basic form), all possible vowel diacritics and 32 high frequency conjuncts. These are the characters shown in black in **Table 1**. The essay contains a total of 104 words or 364 characters. The words used are mostly common and frequently used Bangla words. At this moment, this collection contains 253 copies of this essay, a sample of which is shown in **Fig. 20** (a). These are written by writers of different ages and professions. Each of these documents is stored in two versions: one acquired with a flat-bed scanner at 300 dpi and one digitized with a cellphone camera. All these document pages are manually tagged with associated ground truth of bounding boxes that encapsulate the characters, words and lines from each page. The coordinate information [x_{min}, y_{min}, *height* and *width*] is stored along with the ground truth character, word and line labels in a separate metadata file. To keep access simple, we stored the metadata in plain text (*.txt) files. We used this essay script for both our neural training and testing. The character level ground truth tag data was crucial for us to benchmark the performance of autonomous tagging as described in Section 4.2.1.



(a) (b)

Fig. 20 Samples from the Boise State Bangla Handwriting dataset [2]. (a) Sample essay data and (b) sample conjunct words data with tag overlay of a portion and corresponding recorded metadata.

5.1.2 Conjunct word documents

This part of the Boise State dataset is a page of 106 words containing 128 conjuncts from the Bangla script. These conjuncts are shown in the conjunct column of **Table 1**. The conjuncts were selected by surveying Bangla literature from books, web sites and magazines. Our study shows that the essay script together with the conjunct word document from the Boise State dataset cover 99.7% of the complete Bangla script. The remaining conjuncts which are not included in the dataset are extremely rare. Along with the conjuncts, this page also contains some basic characters and diacritics. There are 70 samples from 70 writers for this document. Each sample is tagged at the character level just like the essay scripts. **Fig. 20** (b) shows a sample conjunct word page and a sample portion with its corresponding ground truth metadata. We used these conjunct word documents both for training and testing our neural networks.

More details about this dataset, such as acquisition process, demographic distribution, preprocessing and formats used can be found in [35]. Although we used three other datasets in our experiments for testing, only this Boise State dataset ensures a robust coverage of the Bangla script and hence we only train our character spotting framework with this dataset.

5.2 External Bangla Datasets

One of the best ways to test the strength and robustness of a recognition system is to test with a dataset which is completely different than the one used for training. With this aim we used three other datasets for testing. These datasets are significantly different from the native Boise State Bangla Handwriting dataset as they are:

1. Collected with different acquisition processes, such as different scanner, settings, pre-processing, paper type, etc.
2. Different in context. They all contain many different words or compositions which our system had never seen during training.
3. Written by an entirely different set of writers with different demographic distribution. Therefore a test with these datasets can ensure our offline recognition framework does not have any bias to a particular type of demographic.
4. Developed in Kolkata, India. Our Boise State dataset is made of contributions from people all from Bangladesh. Although the script is the same, there are differences in handwriting between these two countries. This might not be instantly apparent to most people, but could be substantial in machine learning.
5. One dataset does not contain handwritten documents. It is a collection of historic printed documents - for a rigorous stress-test to our networks.

A sample from each of them is shown in **Fig 21** and the details are presented in the following sections.

5.2.1 CMATERdb 1.1.1

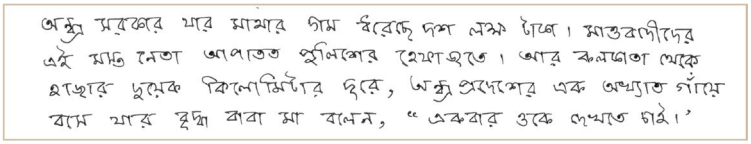
The CMATERdb 1.1.1 is one of the oldest and most used datasets for offline Bangla handwriting research. This contains 100 pages of unconstrained handwritten documents scanned and stored in 24-bit BMP format. Although, there are no transcriptions publicly available for this dataset, the CMATER group provided us with segmented word coordinates for a few of these documents.

5.2.2 Indic Word Dataset

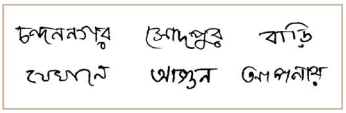
The Indic Word Dataset is a relatively new dataset compared to CMATERdb 1.1.1. This dataset contains segmented Bangla word images, not pages. This also comes with a transcription for each word. Instead of standard Unicode they used Latin counter-forms for Bangla characters, which we converted before using this dataset. This contains 17,091 handwritten word samples with 1,736 unique words. The words are collected from 60 handwritten document images by writers of various professions. We tested our system with the test set from this dataset which contains 3,856 words.

5.2.3 REID2019

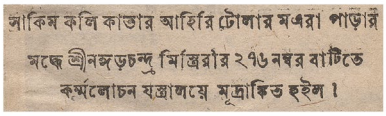
The REID2019 dataset is not an offline handwriting dataset, rather it is a set of scanned historical printed documents used in a conference contest. We used this to test our system's strength. Historical documents suffer from tears and are worn, which results in many different distortions. Additionally, the text content used here is mostly archaic, containing many words rarely used today. Since our approach does not work on any specific word list, this shows how it handles archaic vocabulary. This dataset comes with the metadata necessary to enable us to seamlessly test our networks.



(a) Sample CMATERdb 1.1.1



(b) Sample Indic Word Dataset



(c) Sample REID2019

Fig. 21 Samples from (a) the CMATERdb 1.1.1 [36,13] (b) the Indic Word Dataset [37] and (c) the REID2019 [38] Early Indian Printed Documents dataset.

5.3 Handwritten Korean/Hangul Dataset PE92

For Korean we used the handwritten Korean/Hangul PE92 dataset [34], which is one of the most popular open Korean handwriting datasets. It contains images of 2350 classes of syllables (not Jamos) and about 100 instances of each class. The ground truth is available at the syllable level, but not at the basic character or Jamo level. We used both manual and autonomous tagging on a small subset of this dataset for training K-Net. **Fig 19** shows a few samples that we used from this dataset.

6 Results and Analysis

6.1 Boise State Bangla Handwriting Dataset

For our primary experiment with character spotting and autonomous tagging, we used all 253 essay script documents (both camera-acquired and scanned versions) and 70 conjunct word documents from the Boise State dataset. Word images are extracted from the ground truth information and each image was resized to 600 pixels at its smallest dimension (usually heights) before training. This translates to a total of 60,157 words from which a 90% - 10% training and testing split of the data was made. Words were evenly distributed from the camera-acquired and scanned essay scripts as well as the conjunct word documents. The recognition performance is evaluated with individual mAP and F1 score of C-Net and D-Net as well as Precision, Recall, mAP, F1 score, WRA and CRA of the transcription. We also used an 80% - 20% training and testing combination (similarly evenly distributed) to obtain a 5-fold cross validation result on WRA and CRA.

Table 3 shows all the results of these experiments. The first row presents our earlier results with the character spotting approach [1] for comparison with our current framework. This experiment was based on only 150 camera-acquired essay script documents from the Boise State dataset and did not contain all the conjuncts. The tagging was done manually. We used a 1:3 data augmentation for training, and also used a spell checker after the compilation stage. Details of this experiment can be found in [1].

The second row of **Table 3** shows our current experiment with the whole essay script and conjunct word document dataset where the networks are trained with the precise tag location of the characters. Although the fundamental approach of character spotting is the same in our former experiment, our current setup is based on a much larger training and testing set which also includes the conjunct data. Therefore it covers almost the entire Bangla script unlike the former one which was only based on the basic characters/conjuncts and a few high frequency conjuncts. We did not use any spell checker like our initial experiment, in order to emphasize the raw impact of character spotting approach. We also did not use any data augmentation because, in a way the camera-acquired and scanned sources for the same training data is giving us a 1:2 natural form of data augmentation. With the added amount of data from our first experiment, the recognition

performance was high despite having a larger number of added conjunct classes for the object detection network. In fact, this surpasses the current state-of-art-results for unconstrained Bangla offline recognition. Since we obtained these results without any spell, grammar check or semantic based correction, this has a lot of room for further improvement with careful script-specific postprocessing.

In order to measure the location accuracy with character spotting, we also measured the Intersection over Union or IoU as defined in Eq. (1) with this experiment with precise tagging. We found the average IoU for CNet and D-Net to be 0.53 and 0.47 respectively. Values of IoU around 0.5 are usually considered poor, but in our approach this is expected since we are forcing the networks to be trained with additional diacritic (for CNet) and character (for D-Net) components around the target component as explained in Section 3.1.1. In fact, the C-Net/D-Net detections (orange box) were actually more precise than the auto generated location tags we used for training (blue box) when compared to the actual location of the diacritic (green box) as shown in **Fig 22**. The blue box in this figure contains both a character ‘ ’ and a diacritic ‘ ’ used for both C-Net and D-Net training. Therefore, with this sequential character spotting strategy the IoU scores are expected to be low. However, our aim with character spotting is not to get a precise location since the compilation entirely depends on the relative locations of characters or diacritics.

The last row in **Table 3** shows the results we obtained from the same experimental setup as the second row, but using autonomous tagging instead of precise manual labels. The training used 3 copies of each character data with varying estimated width, η = 20%, 30% and 40% (described in Section 4.2.1). Expectedly, the scores went down a little from what we achieved using precise tags, but the overall process with autonomous tagging is significantly more convenient. To put this into some perspective, the Boise State dataset contains 323 pages of handwritten Bangla documents with approximately 104 words or 364 characters per page. In order to character tag all of these, it took roughly 600 hours of work, equivalent to a full-time job for almost 4 months - all for just one script. This time estimate includes using tools to facilitate the process and excludes the time required for the tool development or data acquisition. Furthermore, it is a fatiguing process to do the tagging manually. Many different kinds of errors like incorrect labeling, duplicate labeling, missing annotation, incorrect annotation size and positioning can happen, which are extremely difficult to spot and fix later. Autonomous tagging, while not relying on precisely perfect annotation, avoids not only this fatiguing process of manual labelling, but also the human errors come along with it. This allows the expansion of the character spotting technique to a new script through on easier, more convenient and much more approachable process. **Figure 23** shows the convenience vs performance of the autonomous tagging approach compared to using precision tagging in order to highlight why this small loss in performance is overly justified.

Table 3 Recognition performance obtained from the character spotting and autonomous tagging approaches with the Boise State Bangla Handwriting dataset. The first row presents our previous achievement with this dataset which uses the same core approach with precision tagging but with a smaller number of detection classes and a smaller train-test set.

Character Spotting Approach	C-Net		D-Net		Transcription						5-fold Cross Validation	
	mAP	F1	mAP	F1	Precision	Recall	mAP	F1	CRA	WRA	CRA	WRA
Reported in [1]	[1]	87.13	89.61	90.34	93.17	88.25	89.42	88.42	89.96	91.09	78.48	-
Precision Tagging	91.41	95.08	92.77	95.38	88.25	91.24	90.32	92.65	94.8	87.74	94.37	86.18
Autonomous Tagging	89.96	92.35	91.25	93.76	-	-	-	-	91.12	80.16	-	-

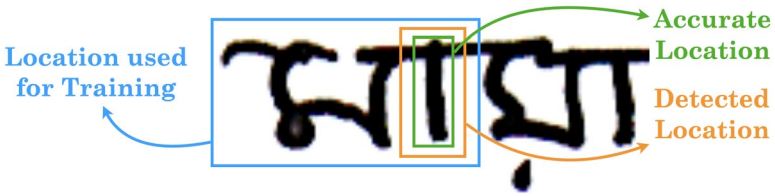


Fig. 22 Demonstration of why IoU with sequential character spotting approach is low. Green shows the exact location of the target diacritic. Blue shows the location used for sequential C-Net/D-Net training which includes the associated consonant with the diacritic. The orange box shows that the D-Net detection is closer to the accurate location than the location used for training.

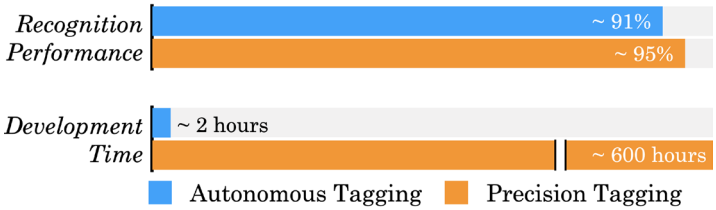


Fig. 23 Comparison of performance and convenience of character spotting approach with precise tagging and autonomous tagging, as observed from the Boise State dataset.

6.2 External Bangla Datasets

To test our framework with external Bangla datasets, we used the networks trained with precision and autonomous tagging from the Boise State dataset and tested on the following:

- CMATERdb 1.1.1:We used the first 25 pages of this dataset with the word coordinates provided by the CMATER group.
- Indic Word Dataset: We used the test set of this dataset which contains 3,856 word images.
- REID2019: The evaluation set from this dataset (contains 56 pages) were used for testing.

Table 4 shows the CRA and WRA obtained from the transcription results. Although the scores are lower than when tested with the Boise State dataset (**Table 3**), it does not deviate much and the total in each case is still a respectable result for datasets not used in training or development. There is no transcription level work reported using the CMATERdb 1.1.1 dataset. The best reported work for the Indic Word Dataset is presented by Mukherjee et al. [37]. They obtained a WRA of 88.19% using a fused LSTM network using a whole word recognition method (explained in Section 2.3) and thereby restricted recognition to only the words available in their dataset. In contrast, our WRA of 78.21% (and 74% with autotag) on this dataset is using a recognition process that is not limited to a fixed set of words. The difference in performance with autonomous tagging from manual precise tagging is consistently low by about 10 - 12%, but again the autonomous tagging method pays off in terms of convenience and ease to add more data.

Surprisingly, the best performance we obtained out of these three datasets is with the REID2019 dataset, which is not handwritten, rather machine printed historical documents. This dataset was used for a competition on recognition of early Indian printed documents at the International Conference on Document Analysis and Recognition (ICDAR) in 2017 and 2019. All the OCR results submitted to this competition were below 80%, while we obtained a CRA of 92.64% and 90.43% with precision and auto-tagging respectively in spite of the fact that our recognition framework has never seen any machine printed text or other image distortion during training. Overall, the outcome of this experiment strongly suggests that our presented framework is robust enough to be used for unconstrained handwriting recognition in real life applications.

Table 4 Recognition performance using character spotting and autonomous tagging with external Bangla datasets.

Dataset Used	Precise Tagging CRA	Precise Tagging WRA	Auto Tagging CRA	Auto Tagging WRA	Reported Max WRA
CMATERdb 1.1.1 [36, 13]	92.36%	82.27%	87.78%	76.64%	N. A.
Indic Words Dataset [37]	89.97%	78.21%	86.12%	74.00%	88.19% [37]
REID2019 [38]	92.64%	81.96%	90.43%	79.10%	< 80% [38]
Boise State Dataset	94.80%	87.74%	91.12%	80.16%	87.74%

6.3 Korean PE92 Dataset

The Korean recognition performance is measured as Jamo Recognition Accuracy or JRA (equivalent to CRA) and compound Syllable Recognition Accuracy or SRA (equivalent to WRA). For training, we used a fraction of the PE92 training set consisting 130 classes of syllables (each class contains 80 to 88 instances) chosen in a manner that every target Jamo shown in **Fig 5** (a) appears relatively evenly. After being trained with autonomously tagged data from this training set, we tested the K-Net performance on the complete PE92 test set which has 2350 classes. We also manually tagged a small portion of the training set (10 samples each from 130 syllable classes) and used that with the same test set to compare the performance. Although the amount of training data used is not equal in these two experiments, the amount of time and effort needed to arrange these was comparable.

The result from recognizing when K-Net was trained with auto and manually tagged characters is presented in **Table 5**, along with two other best scores achieved for this same dataset. For Bangla, we used the same amount of precision tagged and autonomously tagged data and have seen the trained network with precision tagging performs better. However, for Korean we used roughly eight times more autonomously tagged data than precisely tagged data. This mismatch in training data is created to demonstrate what can be achieved within a similar timeframe and/or man-power resource situation with these two approaches. As seen, larger quantity of auto-tag data can outperform a smaller amount of precise-tag trained data, and getting more auto-tagged data is almost effortless where manual-tagging time grows linearly with the amount of data. While [22,23,24] are able to achieve higher recognition rates, they devoted considerably more time to tune their systems for this script and dataset. They also limit their results to 2350 syllable classes, while our approach can operate on any potential combination of the basic Jamos. They used the entire dataset and we only used a fraction of the PE92 dataset for training. Furthermore, unlike Bangla, we did not implement any post processing compilation fixes to assure that the detected combination of Jamos is permissible in the script. Even though vowel position was used in the tag generation process, it was not applied to the detection results. Such post processing steps would very likely improve the recognition performance. From the outcome of this experiment, it can clearly be conjectured that our methodical approach has the potential to achieve a high performing offline recognition rate with small amount of resources like time, human-labor and the amount of data.

Table 5 Korean Recognition Results on PE92 Dataset.

Researchers	Methods	JRA	SRA
Park et al. [22]	MQDF	N. A.	85.99%
Kim et al. [23]	DCNN	N. A.	92.92%
Dziubliuk et al. [24]	Sequence Learning	N. A.	97.90%
Character Spotting	Autonomous Tagging	93.16%	85.52%
	Precision Tagging	84.68%	78.77%

7 Conclusion

We introduced the process of character spotting for unconstrained offline handwriting recognition for Bangla in [1]. Here, first and foremost we upgraded our Bangla recognition with more data and produced a network that can detect all the critical conjuncts of the Bangla script. This produced a fast and high performing offline Bangla recognition framework which works for over 99% of the script without being restricted to any particular set of words. We also omitted the spell correction unit from our original experiment in order to allow more irregular words like names of people or places. This approach is tested robustly with three external Bangla datasets. This is the most successful work reported for unconstrained Bangla (the only other being our previous attempt [1]).

Character spotting is a robust and flexible approach. It is not associated with any specific attribute of a script, and therefore can work for any alphabetic script. Since, most Abugida scripts have close resemblance with Bangla, it is expected the same method will also work for those. Mondal et al. built on our work in [1] to use a similar approach for the Latin script [31] without autotagging. Here we attempted to strengthen our claim of versatility of the character spotting approach with the Korean/Hangul script; a script which is very different from Bangla. We aimed to show how it is to transform of our framework to this script using a small fraction of the PE92 training set. The results we got are already comparable to the state-of-the-art results reported for this dataset, even without applying any simple structure checks.

The biggest bottleneck of the character spotting approach is its core dependency on character level tagging. This requirement makes the development process very complex and time consuming. Data tagging not only takes a lot of time and effort, it is also a very frustrating task to do. Therefore, even if it is almost assured that the framework will work with any alphabetic script, expanding our approach for other scripts is not very practical with the requirement of precise tag training. To fix this, we presented the approach of autonomous tagging, leveraging the fact that our detection networks do not rely heavily on the precise location of a character as long as we can define a bounding box that contains the whole target character. Here, we demonstrated two approaches for autonomous tagging: a sophisticated one for Bangla which estimates the bounding boxes based on the proportions from a computer generated font, and a basic one for Korean which just splits the 2D space in equal heights and/or widths based on the target composition. The autonomous tagging is also flexible and can work for any alphabetic script. Furthermore, this exposes the strength and flexibility of our character spotting approach in whole new way. There are two main ways for advancing this research further: to optimize the system for better performance and to expand it for other scripts.

We believe, the combination of character spotting and autonomous tagging is truly transformational for offline handwriting recognition. It offers a simple and straightforward workflow skipping the delicate and complex processing schemes which were the staples of practice for years. It also offers an open platform which can accommodate many different tools and strategies seamlessly. This technique works great even with a very small amount of data, which is usually not the case with most deep learning approaches. For a new script, our approach only requires a collection of handwritten pages and their transcription. Datasets like this are already available for many scripts like Latin, Arabic, Devanagari, Gurmukhi and Gujarati. When they are not, the process of gathering images of handwritten pages and typing out their content is a small effort compared to other approaches in practice today. Mixed script could be recognized by applying multiple networks to the document. The problem of handwriting recognition has existed for more than half a century and we believe, our presented approach here brings this ancient problem very close to a unified and practical solution.

Acknowledgements

The authors are extremely grateful to all the volunteers who contributed to the Boise State Handwriting Dataset project. We would also like to thank the Center for Microprocessor Application for Training Education and Research (CMATER) group, Dr. Pradeep Kumar, Department of Computer Science & Engineering, Indian Institute of Technology Roorkee, India. Thanks also to Mr. Steven Kim, Department of Computer Science, Boise State University, Boise, Idaho, USA who helped us with Korean. Last, we would like to acknowledge the high-performance computing support of the R2 Compute Cluster provided by Boise State University's Research Computing Department.

References

1. Nishatul Majid and Elisa H Barney Smith. Segmentation-free Bangla offline handwriting recognition using sequential detection of characters and diacritics with a Faster R-CNN. In 2019 International Conference on Document Analysis and Recognition (ICDAR), pages 228–233. IEEE, 2019.
2. Nishatul Majid and Elisa H. Barney Smith. Boise State Bangla Handwriting Dataset. https://doi.org/10.18122/saipl/1/boisestate , 2018.
3. Ethnologue, Languages of the World, (25th ed., 2022). Bengali. https://www.ethnologue.com/language/ben . Online; accessed 24 May 2022.
4. WorldAtlas. The World's Most Popular Writing Scripts. https://www.worldatlas.com/articles/the-world-s-most-popular-writing-scripts.html . Online; accessed 24 May 2022.
5. Samir Malakar, Ram Sarkar, Subhadip Basu, Mahantapas Kundu, and Mita Nasipuri. An image database of handwritten Bangla words with automatic benchmarking facilities for character segmentation algorithms. Neural Computing and Applications, 33(1):449–468, 2021.
6. Poojarini Mitra, Kaustav Bhattacharjee, Anirban Das, Sayan Kumar Dey, Deepjyoti Chakraborty, Aritra Ghosal, and Shadab Akhtar. Character segmentation for handwritten Bangla words using image processing. American Journal of Electronics & Communication, 1(3):8–11, 2021.
7. Monika Kohli and Satish Kumar. Segmentation of handwritten words into characters. Multimedia Tools and Applications, 80(14):22121–22133, 2021.
8. MK Mahto, K Bhatia, and RK Sharma. Robust offline Gurmukhi handwritten character recognition using multilayer histogram oriented gradient features. Int J Comput Sci Eng, 6(6):915–925, 2018.
9. Riya P Javia, Mukesh M Goswami, and Suman K Mitra. Character segmentation from handwritten Gujarati isolated words using deep learning. In 18th India Council International Conference (INDICON), pages 1–6. IEEE, 2021.
10. Deepika Gupta and Soumen Bag. Holistic versus segmentation-based recognition of handwritten Devanagari conjunct characters: a CNN-based experimental study. Neural Computing and Applications, pages 1–17, 2022.
11. Megha Parikh and Apurva Desai. Segmentation of frequently used handwritten Gujarati conjunctive alphabet. In 2019 5th International Conference on Computing, Communication, Control And Automation (ICCUBEA), pages 1–6. IEEE, 2019.
12. Bidyut B Chaudhuri and Abhisek Kundu. Proceedings of the Internation Conference on Frontier in Handwriting Recognition (ICFHR), 2008.
13. CMATERdb: The pattern recognition database repository. http://code.google.com/p/cmaterdb , March 2018.
14. Tapotosh Ghosh, Min-Ha-Zul Abedin, Hasan Al Banna, Nasirul Mumenin, and Mohammad Abu Yousuf. Performance analysis of state of the art convolutional neural network architectures in Bangla handwritten character recognition. Pattern Recognition and Image Analysis, 31(1):60–71, 2021.
15. Mayank Mishra, Tanupriya Choudhury, and Tanmay Sarkar. Devanagari handwritten character recognition. In 2021 IEEE India Council International Subsections Conference (INDISCON), pages 1–6. IEEE, 2021.
16. Manoj Kumar Mahto, Karamjit Bhatia, and Rajendra Kumar Sharma. Deep learning based models for offline Gurmukhi handwritten character and numeral recognition. ELCVIA Electronic Letters on Computer Vision and Image Analysis, 20(2), 2021.

17. N Shobha Rani, AC Subramani, Akshay Kumar, and BR Pushpa. Deep learning network architecture based Kannada handwritten character recognition. In 2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA), pages 213–220. IEEE, 2020.
18. C Vinotheni, S Lakshmana Pandian, and G Lakshmi. Modified convolutional neural network of Tamil character recognition. In Advances in Distributed Computing and Machine Learning, pages 469–480. Springer, 2021.
19. Vijaya Krishna Sonthi, S Nagarajan, and N Krishnaraj. An intelligent Telugu handwritten character recognition using multi-objective mayfly optimization with deep learning based DenseNet model. Transactions on Asian and Low-Resource Language Information Processing, 2022.
20. Bineesh Jose and KP Pushpalatha. Intelligent handwritten character recognition for Malayalam scripts using deep learning approach. In IOP Conference Series: Materials Science and Engineering, volume 1085, page 012022. IOP Publishing, 2021.
21. Vinod Kumar Chauhan, Sukhdeep Singh, and Anuj Sharma. HCR-Net: A deep learning based script independent handwritten character recognition network. arXiv preprint arXiv:2108.06663, 2021.
22. Gyu-Ro Park, In-Jung Kim, and Cheng-Lin Liu. An evaluation of statistical methods in handwritten Hangul recognition. International Journal on Document Analysis and Recognition (IJDAR), 16(3):273–283, 2013.
23. In-Jung Kim and Xiaohui Xie. Handwritten Hangul recognition using deep convolutional neural networks. International Journal on Document Analysis and Recognition (IJDAR), 18(1):1–13, 2015.
24. Valerii Dziubliuk, Mykhailo Zlotnyk, and Oleksandr Viatchaninov. Sequence learning model for syllables recognition arranged in two dimensions. In International Conference on Document Analysis and Recognition, pages 100–111. Springer, 2021.
25. Rahul Pramanik and Soumen Bag. Handwritten Bangla city name word recognition using CNN-based transfer learning and fcn. Neural Computing and Applications, 33(15):9329–9341, 2021.
26. Sandhya Sharma, Sheifali Gupta, Deepali Gupta, Sapna Juneja, Gaurav Singal, Gaurav Dhiman, and Sandeep Kautish. Recognition of Gurmukhi handwritten city names using deep learning and cloud computing. Scientific Programming, 2022.
27. Kartik Dutta, Praveen Krishnan, Minesh Mathew, and CV Jawahar. Offline handwriting recognition on Devanagari using a new benchmark dataset. In 2018 13th IAPR International Workshop on Document Analysis Systems (DAS), pages 25–30. IEEE, 2018.
28. PJ Jino, Kannan Balakrishnan, and Ujjwal Bhattacharya. Offline handwritten Malayalam word recognition using a deep architecture. In Soft Computing for Problem Solving, pages 913–925. Springer, 2019.
29. Dipmala Salunke, Pooja Sabne, Hitesh Saini, Vivekanand Shivanagi, and Pradnya Jadhav. Handwritten Devanagari word recognition using customized convolution neural network. In 2021 International Conference on Computing, Communication and Green Engineering (CCGE), pages 1–5. IEEE, 2021.
30. Chandranath Adak, Bidyut B Chaudhuri, and Michael Blumenstein. Offline cursive Bengali word recognition using CNNs with a recurrent model. In 15th International Conference on Frontiers in Handwriting Recognition (ICFHR), pages 429–434. IEEE, 2016.
31. Riktim Mondal, Samir Malakar, Elisa H Barney Smith, and Ram Sarkar. Handwritten English word recognition using a deep learning based object detection architecture. Multimedia Tools and Applications, pages 1–26, 2021.
32. Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.
33. Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition, pages 248–255. IEEE, 2009.
34. Handwritten Hangul Datasets: PE92, SERI95, and HanDB. https://github.com/callee2006/HangulDB, 1992.
35. Nishatul Majid and Elisa H Barney Smith. Introducing the Boise State Bangla Handwriting dataset and an efficient offline recognizer of isolated Bangla characters. In 16th International Conference on Frontiers in Handwriting Recognition (ICFHR), pages 380–385. IEEE, 2018.
36. Ram Sarkar, Nibaran Das, Subhadip Basu, Mahantapas Kundu, Mita Nasipuri, and Dipak Kumar Basu. CMATERdb1: a database of unconstrained handwritten Bangla and Bangla–English mixed script document image. International Journal on Document Analysis and Recognition (IJDAR), 15(1):71–83, Feb 2011.

37. Subham Mukherjee, Pradeep Kumar, and Partha Pratim Roy. Fusion of spatio-temporal information for Indic word recognition combining online and offline text data. ACM Transactions on Asian and Low-Resource Language Information Processing (TALLIP), 19(2):1–24, 2019.
38. Christian Clausner, Apostolos Antonacopoulos, Tom Derrick, and Stefan Pletschacher. ICDAR2019 competition on recognition of early Indian printed documents– REID2019. In 2019 International Conference on Document Analysis and Recognition (ICDAR), pages 1527–1532. IEEE, 2019.