How to Convert a Latin Handwriting Recognition System to Arabic

Marc-Peter Schambach
Siemens AG
marc-peter.schambach
@siemens.com

Jörg Rottland
Siemens AG
joerg.rottland
@siemens.com

Théophile Alary
Université de
Technologie Troyes
theophile.alary@altheo.net

Abstract

Arabic handwriting recognition systems have been evaluated in public at the ICDAR conferences in 2005 and 2007. This contest provides well defined training data and unpublished test data, which makes the performance of those systems well comparable among each other. The development of the winning system of 2007 is described here in some detail. It is an HMM-based recognition system for Latin script that has been adapted to Arabic script. Only the most fundamental modifications have been applied to the system, like the definition of Arabic character models, and the change of the feature sequences to Arabic writing direction. Other important properties like writing line identification and the features themselves have not been changed. A straightforward optimization process with many small improvements has finally lead to a very competitive high performance Arabic handwriting recognition system. This shows the generality of the selected approach.

Keywords: script word recognition, Arabic handwriting, hidden Markov models, optimization process

1 Introduction

For the second time now, the IfN (Institut für Nachrichtentechnik, TU Braunschweig, Germany) has organized a competition in Arabic Handwriting recognition using its IfN/ENIT database of handwritten Tunisian city names [9]. The results are published at the International Conferences on Document Analysis and Recognition (ICDAR) [7, 6].

For ICDAR 2005, five recognition systems (complemented by one developed by the IfN itself) had been evaluated [7]. See Table 1 for an overview of the 2005 results.

At ICDAR 2007, already 8 groups with 14 systems took part in the contest, proof of the growing interest in Arabic handwriting recognition [1, 2, 3, 8]. The winning system of 2007, that is described in the following, has been developed within short time based on an existing system used for Latin script. In the following sections, the steps are described that have been taken in order to adapt this Latin system to Arabic script. It's a case study of how to efficiently adapt a high performing handwriting recognition system to another alphabet. It has to be noted that none of the authors and developers of the system had real knowledge of Arabic language and script.

Table 1. Recognition results of the 2005 ICDAR competition (set e)

system	best-1	best-5	best-10
ICRA	65.74	83.95	87.75
SHOCRAN	35.70	51.62	51.62
TH-OCR	29.62	43.96	50.14
UOB	75.93	87.99	90.88
REAM	15.36	18.52	19.86
ARAB-IFN	74.69	87.07	89.77

2 Arabic Script

The Arabic alphabet is composed of 28 basic letters. Arabic writing is read from right to left, and top to bottom. Words are segmented into PAWs, "parts of Arabic words", each consisting of a group of letters. A word is composed of one or more PAWs (Fig. 1). There is no difference between upper and lower case nor between written and printed letters. Most of the letters connect directly to the letter which immediately follows, which gives written text an overall cursive appearance.

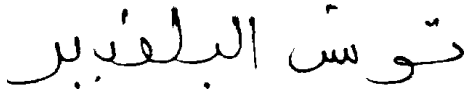


Figure 1. Arabic script sample: "Tunis Belvedere"

Each individual letter can have up to four distinct shapes, dependent on where the letter appears in a word:

- Initial, or medial following a non-connecting letter
- Medial following a connecting letter
- Word-final, following a connecting letter

- Isolated, or word-final following a non-connecting letter

Generally, final and isolated shapes are very similar, such as initial and medial shapes. Some combinations of two or three letters have special shapes called "ligatures", they work exactly as Latin ligatures such as œ and æ.

3 Baseline System

Much of the system is based on techniques developed in 1993 presented in [4, 5]. It's based on Hidden Markov Models, one of the most widespread approaches for cursive script recognition, and can process both cursive and printed script. Over time, a series of improvements has been applied to the system, e.g. [10]. This section shall give a short overview of the system

3.1 Image Preprocessing

Preprocessing operations are performed on the image, the most important steps are [4]:

- Extract the contours as binary connected components. It's a lossless compression and makes the following operations more efficient.
- Smooth the outlines by Wall approximation [11]. This provides further (lossy) compression and is the precondition for further efficient calculation.
- Estimate writing lines. From these lines, rotation and height normalization are deduced.
- Segmentation into frames and zones. The word is cut into small overlapping vertical frames and horizontal zones from which features vectors are extracted.

See Fig. 2 for visualization of these steps.

3.2 Feature Extraction

A small sliding window transforms the preprocessed image into a feature vector sequence, which is interpreted as the emission of the word HMMs. At each window position, about 20 real-valued features for various properties are calculated [4]. See Table 2 for a graphical representation of the features extracted from the window's zones.

3.3 Character Models

Each character is modeled by a separate character-HMM; these are later dynamically composed to word-HMMs. The character model consists of various "paths" with left-to-right topology and self transitions. All paths are jointly followed by an optional pause state. See Fig. 3 for a model with 4 paths and 13 states.

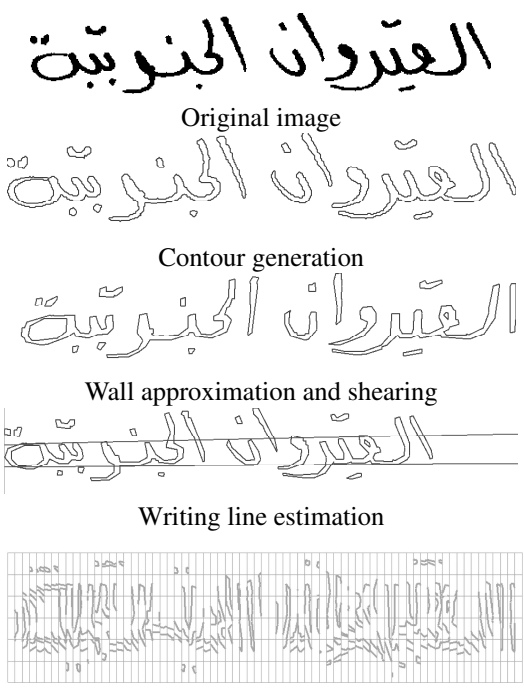


Figure 2. Image preprocessing steps

Each path corresponds to a specific allograph (writing variant) modeling, thus there are as many paths as there are allographs of a character. In Latin script, allographs that are typically modeled are upper and lower case, handprint and cursive.

4 Basic Adaptations

The initial step in adaptation to Arabic has been to make the system run. The existing system had not been designed to process Arabic words, and as a first step, its performance without major modifications had to be evaluated. To achieve this, only the most essential adaptations had been performed.

Table 2. Two sets of features extracted from the sliding window's 6 zones. See [4] for details.

Zone	set 1	set 2
Top	. |_	|
XEnd	|_ / \	|_ / \
Upper Center		|_ / \
Center	( ) |_ / \	|_ / \
Base	∨ |_ / \	∨ |_ / \
Bottom	|	|

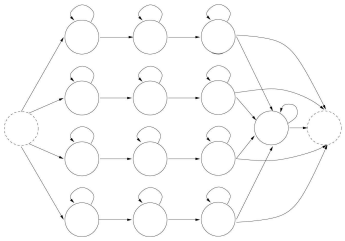


Figure 3. Topology of a character HMM, each path representing an allograph.

4.1 Preprocessing

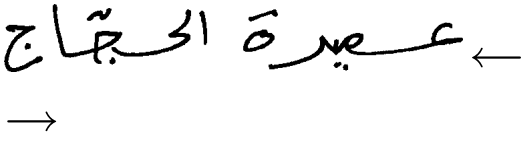
The biggest difference from Latin to Arabic script is writing direction. Therefore, the direction of feature extraction had to be reverted. The two straightforward possibilities, feature sequence reordering and lexical string inversion, have been implemented.

No other adaptations have been made. Especially has to be noted that *no* efforts are spent on detecting PAWs.

4.2 Character Encoding

The next step was to define the Arabic character modeling. The model definitions had to be adapted to the encoding used by IfN/ENIT.

Since the baseline system was designed for Latin characters, there are alternative writing styles allowed for each letter, intended to allow upper/lower-case and printed/cursive script distinction. As in Arabic there are also four distinctions for each letter, depending on their location within a word, these four variants could be directly matched to Latin counterparts. A phonetically inspired transliteration similar to Buckwalter transliteration [12] has been chosen which allows direct usage of the existing training and testing tools.



ayB|maM|yaM|raE|teA|aaA|haMlaB|jaMllL|aaE|jaA|

(E) ˜(M) ˜(|Y) ˜(r) (p) (a) (=) ˜(J) ˜(a) (j)

Figure 4. Example of Buckwalter inspired transliteration: top: IfN label, bottom: modified Buckwalter label, with upper/lower case and tilde (˜) specifying writing style. The arrows indicate writing direction.

Fig. 4 shows an example of transliteration from the IfN/ENIT label to a “Latin-like” label. It also shows two types of *ligatures*: a multiple character ligature “haMlaB” and a chadda ligature “jaMllL”.

4.3 Parameter Adaptation

The HMM structure of the character models has been kept, with four writing variants for each character, as in Latin script (Fig. 3). Parameter adaptation has been performed by standard Baum-Welch training, an iterative expectation-maximization (EM) algorithm. Training has been performed with the data sets a-d which had been published for the 2005 contest, tests have been done mostly on data set e which has been published after 2005 only. The first results (Table 3) are very promising already.[1]

5 Advanced Adaptations

Brute force approaches have now been used to adapt some of the parameters. Most potential for improvements of the recognition engine has been seen in:

- Writing line estimation and feature extraction.
- Correct and useful modeling of Arabic characters and its variants.
- Identification of ”simple” errors, e.g. wrong labels.

Many experiments have been performed to find an optimal configuration. For each experiment, a full training has been executed each time preprocessing configurations or label transliterations have been changed. (Only the affected parts of the system have been re-trained.)

5.1 Preprocessing Parameters

The preprocessor extracts features from images (section 3.1). Many parameters have been candidates for optimization; only the most promising have been chosen which leaves room for further improvements. *Not* adjusted have been, for example, weights for baseline estimation.

5.1.1 Window Width

Feature generation uses a sliding window, whose width and overlap can be adjusted (Fig. 5). A window too narrow may result in the content not be properly analyzed, while a window too wide may cause incorrect character segmentation.

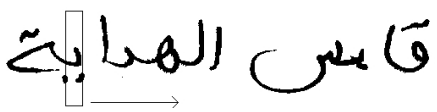


Figure 5. Image sampling using a sliding window

[1] The various experiments seem to make inconsistent use of the IfN/ENIT database sets a-e. However, this reflects the fact that data set e got available to developers only during project run time. In addition, the experiments reported here are not in strict chronological order.

Tests with various window widths from 7 to 13 pixels, have been performed, with best results at 11 pixels (see Table 3). The overlap has always been set at 2/3 of the window width, some other values have been tested without success.

Table 3. Recognition rates with variable window widths (set d, training on sets a-d)

Width	1-best	5-best	10-best
8	81.38	91.33	93.30
9	82.23	92.30	94.18
10	83.83	93.53	95.00
11	**84.72**	**94.18**	**95.69**
12	83.45	93.33	95.18
13	84.10	93.08	95.06

5.1.2 Writing Line Estimation

The writing lines bound the global shape of the writing. They are required to perform correct rotation and height normalization. The preprocessor can use different configurations giving various writing line alternatives. The type of features (Table 2) also depends on these configurations.

5.1.3 Reading Direction

Both inversion methods for writing direction (section 4.1) have been tested. Character inversion (reading from left to right) gives worse results, maybe due to pruning during Viterbi trie evaluation which works best from word beginning.

5.2 Ligature Modeling

5.2.1 Standard Ligatures

The references labels of the IfN/ENIT database contain 10 ligatures which occur more than 4000 times within the database. Arabic ligatures differ strongly from the combination of the independent characters (Fig. 6). This makes their independent modeling a key issue for recognition performance (Table 4).

Ligature (khMlaB) (laB) and (khM) without ligature

Figure 6. Arabic ligatures examples (IfN/ENIT labels)

5.2.2 Chadda Ligatures

The ligature *chadda* is special; it looks like a small *w* above a character and is used to accentuate the consonant below. Size of the chadda ligature is very variable, it may be as big as the character below (Fig. 7). It is present at many characters.

Table 4. Influence of ligature modeling on recognition performance (set d, training on sets a-d)

Experiment	1-best	5-best
4 ligatures implemented	89.58	96.53
All ligatures implemented	**90.77**	**96.93**

Figure 7. Three styles to write the chadda ligature

As many characters may occur with chadda ligature, new models for "chadda-ligatured" characters are used: Out of 40 character models, a set of additional 20 models with chadda has been created. Recognition rate increases (Table 5), but computation time also grows since the number of writing variants within the lexicon grows from 1300 to almost 1700 words. Additionally, with many special chadda models the risk of over-training arises. Thus cross-validation has been performed, and only those characters which frequently appear with chaddas in the training data have finally been modeled .

Table 5. Influence of chadda ligature modeling (training on sets a-e)

Experiment	set d	set e
Without chadda ligature	91.26	83.36
With chadda ligatures (over-trained)	**93.05**	**83.91**

5.3 Character Modeling

5.3.1 Character Merging

Some Arabic characters have very similar appearance, thus we can suppose that merging them into a single model can increase recognition performance by more discriminative training and avoiding permutation errors with similar letters (see Tab. 6). Additionally, it allows a reduced number of writing variants within the lexicon, which results in faster computation.

5.3.2 Number of Allographs

Arabic characters may have up to four allographs depending on their location within the word. Normally, one character HMM path (Fig. 3) is used for each allograph,

Table 6. Recognition performance with merged models (set e)

Training sets	abcd	abcde	abcde	abcde
Pruning factor	1.0	1.0	1.3	1.5
Reference	**81.89**	83.08	83.36	83.33
5 models merged	80.28	**83.23**	**83.44**	**83.51**

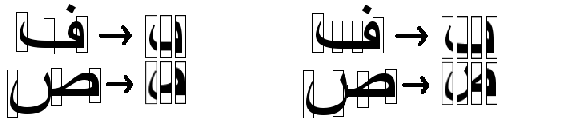
but fewer allographs may be modeled if shapes are similar. The best results are obtained if all allographs are modeled, even when shapes are very similar. However the gain is very low (Table 7).

Table 7. Influence of the number of modeled allographs on recognition rate.

Experiment	set d	set e
Manual selection of allographs	90.77	82.93
All allographs modeled	**90.94**	**83.08**

5.4 Model Length Adaptation

More recently implemented (not included in the 2007 contest release) is adaptive modeling of the HMM model duration [10]. All models are built with variable length of the allograph paths, representing the complexity of the shapes. Since a lot of wide characters exist in Arabic writing, this feature proved to be effective (Fig. 8). Increase of recognition rate of about 1% has been measured.



Fixed length With variable length adaptation

Figure 8. With variable length model adaptation, models better represent the character's complexity

5.5 Pruning

Pruning is used in order to speed up the recognition process: For recognition a lexicon trie is built; during Viterbi decoding, only the locally most probable branches are kept. The "pruning factor" defines the minimal probability for a branch to be kept. It strongly affects computation time (a factor of 2 between 1.0 and 1.3!). However, a too low pruning factor may cause the globally best result to be lost, but experiments show, that it also may *improve* the recognition rate when optimal but wrong alternatives are pruned (Table 8).

Table 8. Influence of pruning factor on recognition rate (set e). The absolute value of the pruning factor is implementation specific.

Pruning factor	1.0	1.2	1.3	1.5	4.0
Computation time	fast				slow
Results	83.08	83.28	**83.36**	83.33	83.33

6 Voting

Voting is used to combine the strengths of various recognition engines. Also a low performing recognizer can positively contribute, if it's results are independent of the other recognizers. However, using a voter is time-consuming. Experiments with up to three recognizers have been performed.

6.1 Maximum Rule

With maximum rule, the recognizer with the highest result credibility wins. For two preprocessing configurations, using different writing line estimations, tests have been performed which show significant performance gains (Table 9).

Table 9. Maximum rule voting between different preprocessing configurations (set e, training on sets a-e)

Preprocessing	1	2	1+2 voted
1-best	69.94	83.28	**84.87**
5-best	84.57	**93.65**	92.82

6.2 Weighted Sum Voting

Standard sum voting adds all credibilities for each result alternative; the alternative with the highest sum wins. This method gives even better results than maximum rule (Table 10). Results can be raised by further 0.4% with "weighted sum voting", where summands are weighted by recognition performance. Finally, a voting combination of three recognizers which differ in writing line estimation (5.1.2) and reading direction (5.1.3) has been chosen.

Table 10. Voting rule influence on recognition results (set d, training on sets a-e; voting 3 configurations)

	Maximum rule	Sum voting
1-best	93.76	**94.58**
5-best	98.11	**98.75**

7 Results

Two systems have finally been submitted to the contest: A *fast* system, using the single best standalone HMM recognizer, and the *best* system, that merges the results of

three different HMM recognizers using weighted sum voting. Both have been trained with data sets a-e only, no further training data has been used. Table 11 shows detailed results which have been measured by IfN on the submitted systems [6]. The results of all contest participants are listed in Table 12.

Table 11. Final recognition results (training on sets a-e; sets f and s are unreleased.)

	set d	set e	set f	set s	time(ms)
fast	91.23	84.27	82.77	68.09	39.2
best	94.58	88.41	87.22	73.94	109

Table 12. ICDAR 2007 competition results (set f, only the best system of each group)

system	best-1	best-5	best-10
MITRE	61.70	81.61	85.69
CACI(3)	14.78	29.88	37.91
CEDAR	59.01	78.76	83.70
MIE	83.34	91.67	93.48
SIEMENS(2)	**87.22**	**94.05**	**95.42**
UOB-ENST(3)	81.93	91.20	92.76
ICRA	81.47	90.07	92.15
PARIS V	80.18	91.09	92.98

The recognition rate of the initial system (section 4) has been around 83% (Table 3) and has been improved by parameter tuning (section 5) and voting (section 6) to more than 94% on set d. The system doesn't seem to be over-trained as results on the unknown set f are comparable to those on the known set e. The recognition rate is still low on the unpublished set s, which is from writers from another country and considered to be more difficult.

8 Conclusions

Some reasons may be given how this system could achieve such a good performance despite the fact that no domain knowledge has been implemented and only limited efforts have been spent on it.

First, it shows that the described HMM based script word recognition method has a high generalization degree to adapt it with little effort to any other alphabet with similar structure.

An explanation may also be recognition speed. It can often be read, that for the development of a system the priority is focused on recognition performance and not on computation time. Basically this is implied by the idea to first find a working algorithm and to improve its runtime later on after it proved to be working. This approach has a major drawback - time is a limited resource for a developer. Because our system is quite fast, we were able to perform a lot of experiments during development, resulting in a much better system than we would have gotten with a slower system.

Another reason may be the limited task of the contest. In general for Arabic recognition, the correct handling and recognition of dots above and below the characters is as important as it is difficult. However, in the IfN/ENIT database of Tunisian town names the dots are not as important: Even if they were completely ignored, less than 1% of the dictionary entries would become ambiguous. Thus in this test the dot recognition has an untypical low importance for Arabic word recognition.

Potential for further increase of recognition rates is seen in more elaborate chadda modeling (section 5.2.2) and model length adaptation (section 5.4).

References

[1] A. Abdulkader, "Two-Tier Approach for Arabic Offline Handwriting Recognition", *Proc. of the 10th IWFHR*, Oct. 2006, La Baule, France.

[2] H. E. Abed and V. Märgner, "Comparison of Different Preprocessing and Feature Extraction Methods for Offline Recognition of Handwritten Arabic Words", *Proc. of the 9th ICDAR*, 2007, pp 974–978, Curitiba, Brazil.

[3] R. Al-Hajj, C. Mokbel and L. Likforman-Sulem, "Combination of HMM-Based Classifiers for the Recognition of Arabic Handwritten Words", *Proc. of the 9th ICDAR*, 2007, pp 959–963, Curitiba, Brazil.

[4] T. Caesar, J. Gloger and E. Mandler, "Preprocessing and Feature Extraction for a Handwriting Recognition System", *Proc. of the 2nd ICDAR*, Oct. 1993, pp 408–411, Tsukuba Science City, Japan.

[5] A. Kaltenmeier, T. Caesar, J. Gloger and E. Mandler, "Sophisticated topology of hidden Markov models for cursive script recognition", *Proc. of the 2nd ICDAR*, Oct. 1993, pp 139–142, Tsukuba Science City, Japan.

[6] V. Märgner and H. E. Abed, "ICDAR 2007 — Arabic Handwriting Recognition Competition", *Proc. of the 9th ICDAR*, 2007, pp 1274–1278, Curitiba, Brazil.

[7] V. Märgner, M. Pechwitz and H. E. Abed, "ICDAR 2005 — Arabic Handwriting Recognition Competition", *Proc. of the 8th ICDAR*, 2005, pp 70–74, Seoul, Korea.

[8] F. Menasri, N. Vincent, E. Augustin and M. Cheriet, "Shape-based Alphabet for Off-line Arabic Handwriting Recognition", *Proc. of the 9th ICDAR*, 2007, pp 969–973, Curitiba, Brazil.

[9] M. Pechwitz, S. S. Maddouri, V. Märgner, N. Ellouze and H. Amiri, "IFN/ENIT-database of handwritten Arabic words", *Proc. of CIFED*, 2002, pp 129–136, Hammamet, Tunisia.

[10] M.-P. Schambach, "Model Length Adaptation of an HMM based Cursive Word Recognition System", *Proc. of the 7th ICDAR*, Aug. 2003, Edinburgh, Scotland.

[11] K. Wall and P.-E. E. Danielsson, "A Fast Sequential Method for Polygonal Approximation of Digitized Curves", *Computer Vision, Graphics, and Image Processing*, 28(2):220–227, 1984.

[12] Wikipedia"Buckwalter transliteration — Wikipedia, The Free Encyclopedia", `http://en.wikipedia.org/wiki/Buckwalter_Transliteration`, 2008.