PAPER

# Development of a Robust and Compact On-Line Handwritten Japanese Text Recognizer for Hand-Held Devices

**Jinfeng GAO**[†a)], ***Student Member***, **Bilan ZHU**[†], ***Member***, ***and*** **Masaki NAKAGAWA**[†], ***Fellow***

**SUMMARY** The paper describes how a robust and compact on-line handwritten Japanese text recognizer was developed by compressing each component of an integrated text recognition system including a SVM classifier to evaluate segmentation points, an on-line and off-line combined character recognizer, a linguistic context processor, and a geometric context evaluation module to deploy it on hand-held devices. Selecting an elastic-matching based on-line recognizer and compressing MQDF2 via a combination of LDA, vector quantization and data type transformation, have contributed to building a remarkably small yet robust recognizer. The compact text recognizer covering 7,097 character classes just requires about 15 MB memory to keep 93.11% accuracy on horizontal text lines extracted from the TUAT Kondate database. Compared with the original full-scale Japanese text recognizer, the memory size is reduced from 64.1 MB to 14.9 MB while the accuracy loss is only 0.5% from 93.6% to 93.11%. The method is scalable so even systems of less than 11 MB or less than 6 MB still remain 92.80% or 90.02% accuracy, respectively.

***key words:*** *on-line recognition, handwritten text recognition, elastic matching, MQDF, vector quantization*

## 1. Introducton

With the development of pen-based or touch-based hand-held devices, handwritten text recognition system running on such hand-held devices needs to be developed. The relatively small RAM of a hand-held device requires a handwritten text recognition system as small as possible that maintains high accuracy. This paper focuses on constructing a compact on-line handwritten Japanese text recognition system, running on such devices.

Handwritten character recognition mainly includes two types of methods: on-line and off-line recognition. The on-line method works on an input on-line handwritten character pattern, which is a time-sequence of pen-tip coordinates, and although it is easily made robust to stroke connection and deformation, it is sensitive to stroke order variation. On the other hand, the off-line method recognizes an off-line pattern, which is a character pattern image, and although it is insensitive to stroke order variation or duplicated strokes, it is not very robust to stroke connection and deformation. To overcome the disadvantage of the on-line method, the off-line recognition method is combined with the on-line method to form a combined recognizer since the off-line method is made applicable to an on-line pattern by discarding time-sequence information [1].

Furthermore, to improve the accuracy of text recognition, text recognition systems integrate over-segmentation, character recognition, linguistic context evaluation and geometric context evaluation [2], [3]. Moreover, Japanese is a large character set language that uses thousands of ideographic characters of Chinese origin, two sets of phonetic characters, alpha, numerics and symbols, so designing a compact yet robust text recognition system running on hand-held devices is challenging. We need to compress each component in the text recognizer while keeping high accuracy.

For Japanese and Chinese off-line handwritten character recognition, MQDF2 [4] has been widely used, but its performance depends on two parameters: the size of feature vector dimensions and the number of principal components. The off-line recognizer with different parameter configurations can obtain different performances of character recognition with different memory costs and recognition speeds. In practice, a configuration of 160 dimensions and 40 principle components for MQDF2 is widely used to decrease the memory cost and keep the accuracy high. However, about 90 MB is required for the configuration covering 7,097 character classes that consist of 6,715 Kanji (ideographic characters) of the 1st set and the 2nd set in the Japanese Industrial Standard (JIS) and 382 characters from Kana (phonetic characters), symbols, alpha, numeric and Greek letters. This is too large for hand-held devices, so its compression plays a large role in construction of a compact text recognizer.

Liu et al. [5] used LVQ and converted 4-byte parameters into 1-byte to build a compact classifier for printed character recognition of a large character set. Because handwritten text recognition is much more challenging than printed character or text recognition, this approach alone cannot ensure high accuracy of handwritten text recognition. Although Long et al. built a compact MQDF2 recognizer by vector quantization [6], it just focuses on character recognition rather than text recognition and there are only 3,777 character classes.

As for on-line recognition, to achieve elastic matching between feature points, DP-matching was introduced [7], [8]. Thus, we proposed its greedy variation named Linear-time Elastic Matching, which was quicker than DP-matching with deterministic matching and limited backtracking. LTM was combined with structured character pattern representation where radical patterns were shared among character patterns with the effect of memory reduction and consistency against style variations [9].

Manuscript received July 27, 2012.
Manuscript revised October 30, 2012.

[†]The authors are with the Graduate School of Engineering, Tokyo University of Agriculture and Technology, Koganei-shi, 184–8588 Japan.
a) E-mail: gaojinfeng19790213@yahoo.co.jp

DOI: 10.1587/transinf.E96.D.927

HMM succeeded DP-matching in on-line handwriting recognition just as in speech recognition [10]. Recently, we have shown that Markov Random Field (MRF) provides better recognition accuracy than LTM or even HMM [11]. Although MRF is more accurate, it has larger memory cost than LTM. Typically, for 7,097 character classes, the memory cost of MRF is about 20 MB, but LTM just needs less than 300 KB. If the on-line and off-line combined recognizer is used, LTM is the better choice.

In our previous work [12], we also constructed a compact handwritten Japanese text recognition system by selecting a small off-line recognizer to optimize tradeoff between accuracy and memory-time cost. In this paper, to deploy the compact text recognizer on hand-held devices, we will further compress each component of the text recognition system. Specifically, the most important and challenging compression on the combined character recognizer of integrated text recognition system is successfully done by selecting LTM for on-line recognition and compressing MQDF2 via a combination of LDA, vector quantization, and data type transformation. The system has been pre-installed into a type of smart phone.

The rest of this paper is organized as follows: Sect. 2 describes model of text recognition. Section 3 explains the architecture and design of the text recognizer. Section 4 describes pattern databases and benchmark. Section 5 presents the construction of the compact text recognizer. Section 6 reports experimental results and analysis. Finally, we draw the concluding remarks.

## 2. Model of Text Recognition

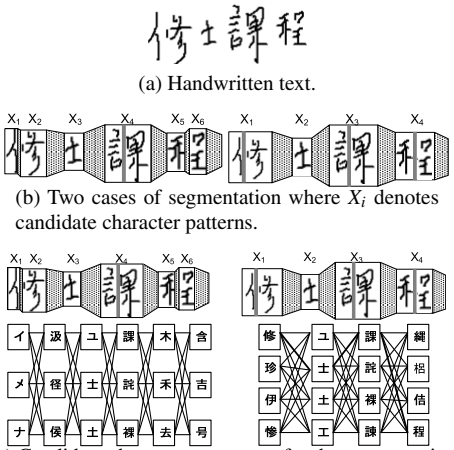
Given a handwritten text $T$, the probability that $T$ is segmented to a sequence of candidate character patterns $X = x_1 x_2, \cdots, x_m$, and a candidate character pattern $x_i$ is recognized as a character class $c_i$, forming a recognized text string $C = c_1 c_2, \cdots, c_m$, is defined as the conditional probability $P(C, X|T)$ and is transformed as follows:

$$P(C, X|T) = \frac{P(C) \times P(T, X|C)}{P(T)}. \tag{1}$$

The goal is to find the candidate character pattern sequence $X$ and the character sequence $C$ that maximize $P(C, X|T)$ among candidate segmentations as shown in Fig. 1 (b) and among candidate character sequences as shown in Fig. 1 (c). Since $P(T)$ is the probability that a handwritten pattern T occurs regardless of $X$ and $C$, we ignore it. Hereafter, we will consider $P(C)$ and $P(T, X|C)$. Since character patterns cannot be reliably segmented without being recognized so that we follow the segmentation by recognition strategy, i.e. we try to find the best segmentation and recognition to maximize $P(C)P(T, X|C)$.

Further, since a sequence of candidate pattern $X$ is generated from $T$, we can assume $P(T|X) = 1$. Then, obtain the Eq. (2).

$$P(C)P(T, X|C) = P(C)P(X|C)P(T|X) = P(C)P(X|C) \tag{2}$$



(a) Handwritten text.

(b) Two cases of segmentation where $X_i$ denotes candidate character patterns.

(c) Candidate character sequences for the segmentations in (b).

**Fig. 1** Segmentation and recognition.

The probability $P(C)$ is evaluated by tri-gram linguistic model and we assume that the feature vectors $b_i$, $q_i$, $x_i$, $p_i^u$, $p_i^b$ and $g_j$ which are used for segmentation and recognition are independent of each others, then Eq. (3) is obtained.

$$P(C)P(X|C) = \prod_{i=1}^{m} \begin{matrix} P(c_i|c_{i-2}c_{i-1})P(b_i|c_i)P(q_i|c_i)P(p_i^u|c_i) \\ \times P(x_i|c_i)P(p_i^b|c_{i-1}c_i)P(g_i|c_i) \end{matrix} \tag{3}$$

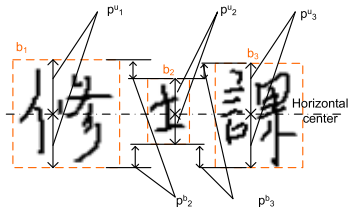
However, the probability formulation (3) is still insufficient because it does not consider different contributions of the different feature vectors in segmentation and recognition. We take the logarithm of probability and incorporate the weights of different feature vectors to get a generalized evaluation model function $f(X, C)$ expressed as Eq. (4). $t_i$ denotes Sb or Sw which is described later.

$$f(X, C) = \sum_{i=1}^{m} \begin{pmatrix} \lambda_1 \log P(c_i|c_{i-2}c_{i-1}) + \lambda_2 \log P(b_i|c_i) \\ +\lambda_3 \log P(q_i|c_i) + \lambda_4 \log P(p_i^u|c_i) \\ +\lambda_5 \log P(x_i|c_i) + \lambda_6 \log P(p_i^b|c_{i-1}c_i) \\ +\lambda_7 \log P(g_i|t_i) \end{pmatrix}. \tag{4}$$

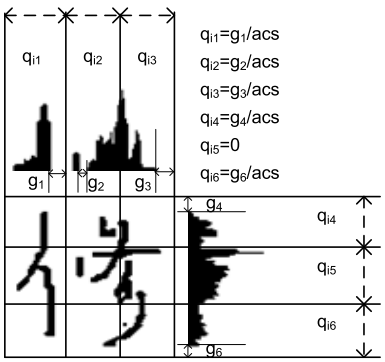
By assigning different weights of start segment and non-start segment based on their different contributions to segmentation and recognition, we obtain Eq. (5). The detail about the evaluation model is originally described by Zhu et al. [13].

$$f(X, C) = \sum_{i=1}^{m} \begin{pmatrix} \sum_{h=1}^{6} [\lambda_{h1} + \lambda_{h2}(k_i - 1)] \log p_h \\ +\lambda_{71} \log P(g_{j_i}|Sb) \\ +\lambda_{72} \sum_{j=j_i+1}^{j_i+k_i-1} \log P(g_j|Sw) \end{pmatrix} + m\lambda. \tag{5}$$

where $P_h$ ($h = 1, \cdots, 6$) stand for the probabilities of $P(c_i|c_{i-2}c_{i-1})$, $P(b_i|c_i)$, $P(q_i|c_i)$, $P(p_i^u|c_i)$, $P(x_i|c_i)$ and $P(p_i^b|c_{i-1}c_i)$, respectively. Moreover, $P(g_{j_i}|Sb)$ is the probability that spacing between character patterns (Sb) appears




(a) $b_i$, $p_i^u$ and $p_i^b$ feature vectors.




(b) inner-gap feature vector $q_i$.

**Fig. 2** Geometric features of a candidate character pattern.

as $g_{j_i}$ and $P(g_j|Sw)$ is the probability that spacing within a character pattern (Sw) rather than between character patterns appears as $g_j$. Instead of $P(g_{j_i}|Sb)$, $P(g_{j_i}|Sb, c_{i-1}, c_{i+1})$ is more precise, but we simply omit the effect of neighboring characters. $k_i$ denotes the number of blocks separated by Sw's within a candidate character pattern $x_i$.

$P(c_i|c_{i-2}c_{i-1})$ is tri-gram linguistic context probability. A smoothed tri-gram linguistic context model by interpolation of uni-gram, bi-gram and tri-gram is used to evaluate linguistic context $P(c_i|c_{i-2}c_{i-1})$ as given by Eq. (6), where $\lambda_1 = 0.7$, $\lambda_2 = 0.2$ and $\lambda_1 = 0.1$ are used.

$$P_s(c_i|c_{i-2}c_{i-1}) = \lambda_1 \times P(c_i|c_{i-2}c_{i-1}) + \lambda_2 \times P(c_i|c_{i-1}) + \lambda_3 \times P(c_i). \quad (6)$$

The terms in Eq. (5), $b_i$, $q_i$, $p_i^u$ and $p_i^b$ are geometric feature vectors derived from a candidate character pattern in handwritten text as shown in Fig. 2. The term $b_i$ is composed of the height and width of the bounding box of a candidate character pattern. The term $p_i^u$ consists of two vertical distances from the horizontal central line of a text line to the top and bottom of the bounding box. The term $p_i^b$ is composed of a vertical distance between the top edges of the bounding boxes of two adjacent candidate character patterns in a text line and that between the bottom edges of the bounding boxes. The term $q_i$ is an inner-gap vector in a candidate character pattern, which is obtained by projecting the character pattern into the horizontal and vertical directions, splitting each of their histograms into three slices, finding a gap or gaps in each slice, summing total lengths of gaps, and dividing it by the estimated average character size noted as *acs* as shown in Fig. 2 (b).

We assume $P(b_i|c_i)$, $P(q_i|c_i)$, $P(p_i^u|c_i)$ and $P(p_i^b|c_{i-1}c_i)$ to be normal distributions and model their logarithms by a quadratic discriminant function (QDF), which can be trained by training patterns. They are together called geometric context. On the other hand, $P(p_i^b|c_{i-1}c_i)$ requires large memory cost, since it is two dimensional.

The term $g_j$ denotes another geometric feature vector concerning segmentation point, which is extracted from an off-stroke and its surroundings. We call it especially a spacing feature vector. Table 1 lists 18 elements of $g_j$ and Table 2 lists features to obtain them. Here, a bounding box is determined from the top-left and the bottom-right coordinates of a stroke or strokes. Details are described by Zhu and Nakagawa [14].

The remaining probability $P(x_i|c_i)$ in Eq. (5) is evaluated by the character recognizer that is described in Sect. 3.2.2.

**Table 1** Elements of a spacing feature vector.

| Element | Definition | | Element | Definition |
|---|---|---|---|---|
| $f_1$ | $T_{off}$ | | $f_{10}$ | $O_B$/area of $B_s$ |
| $f_2$ | $D_{B\_y}$/acs | | $f_{11}$ | $O_B$/ area of $B_p$ |
| $f_3$ | $O_{BA}/acs^2$ | | $f_{12}$ | $O_B/acs^2$ |
| $f_4$ | $D_{B\_x}$/width of $B_p$ | | $f_{13}$ | $DB_x$/acs |
| $f_5$ | $D_{B\_x}$/width of $B_s$ | | $f_{14}$ | $DB_y$/acs |
| $f_6$ | $D_{B\_x}$/acs | | $f_{15}$ | $DB_A$/acs |
| $f_7$ | $D_{B\_y}$/height of $B_p$ | | $f_{16}$ | $DA_s$/acs |
| $f_8$ | $D_{B\_y}$/height of $B_s$ | | $f_{17}$ | $L_{off}$/acs |
| $f_9$ | $D_{B\_y}$/acs | | $f_{18}$ | sine($L_{off}$) |

**Table 2** Features to obtain the spacing feature vector.

| Feature | Definition |
|---|---|
| $B_p$ | Bounding box of the immediate preceding stroke |
| $B_s$ | Bounding box of the immediate succeeding stroke |
| $B_{p\_all}$ | Bounding box of all the preceding strokes |
| $B_{s\_all}$ | Bounding box of all the preceding strokes |
| $D_{B\_x}$ | Distance between $B_s$ and $B_p$ in x-axis |
| $D_{B\_y}$ | Distance between $B_s$ and $B_p$ in y-axis |
| $O_{BA}$ | Overlap area between $B_p$ and $B_s$ |
| $DB_x$ | Distance between centers of $B_s$ and $B_p$ in x-axis |
| $DB_y$ | Distance between centers of $B_s$ and $B_p$ in y-axis |
| $DB_A$ | Absolute distance of centers of $B_p$ and $B_s$ |
| acs | Average character size of text line |
| $D_{BA\_s}$ | Vertical distance between $B_{p\_all}$ and $B_s$ |
| $T_{off}$ | Time lapse of the off-stroke |
| $L_{off}$ | Length of the off-stroke |

## 3. Architecture and Design of Text Recognizer

This section describes the architecture and design of the handwritten text recognizer.

### 3.1 Handwritten Text Recognizer

Figure 3 shows the block diagram of the handwritten text recognizer. First, input handwritten text composed of a sequence of strokes is over-segmented into a sequence of blocks (primitive segments) as shown in Fig. 4 (a). In more detail, every off-stroke (pen lift between two consecutive strokes) is classified by SVM into segmentation point (SP), non-segmentation point (NSP) or undecided point (UP) according to the features such as distance and overlap between

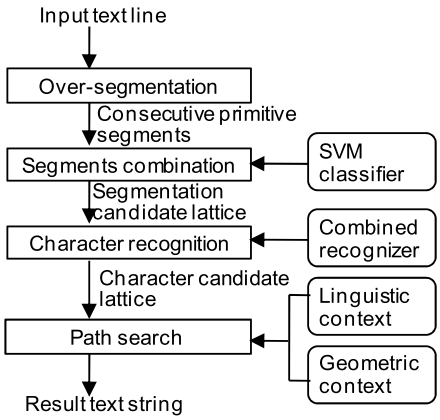
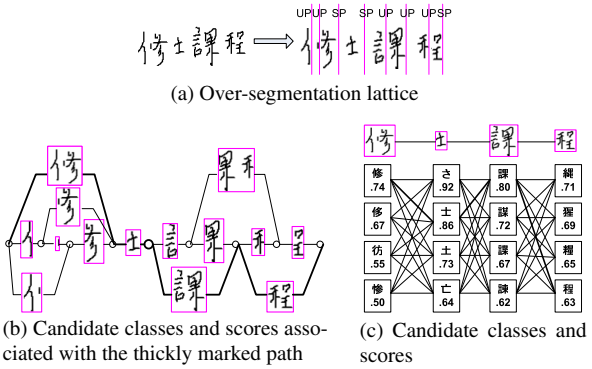



**Fig. 3** Architecture of handwritten text recognizer.




(a) Over-segmentation lattice

(b) Candidate classes and scores associated with the thickly marked path

(c) Candidate classes and scores

**Fig. 4** Over-segmentation and csr-lattice.

consecutive strokes, where UP is treated either as SP or NSP [14]. We must consider that UP can be either a boundary between character patterns or within a character pattern, so we represent possible segmentations as a lattice as shown in Fig. 4 (b) where primitive segments and consecutive primitive segments beside UP form candidate character patterns marked by boxes. Concatenation of consequent primitive segments is limited by their total lengths.

Then, each candidate character pattern is recognized by the character recognizer, and several pairs of a character recognition candidate (class) and its score are associated with each candidate character pattern. We call a candidate segmentation lattice with candidate classes and scores associated with each candidate character pattern a candidate segmentation recognition lattice, or csr-lattice for short. Since adding candidate classes and scores into Fig. 4 (b) complicates it, we use Fig. 4 (c), which shows candidate classes and scores to the candidate character patterns along the thickly marked path in Fig. 4 (b).

Finally, the task of text recognition is to find the optimal path in the csr-lattice. We use the beam search strategy. Considering that character patterns cannot be reliably segmented without being recognized due to the irregularity of character size and spacing, to ensure high accuracy of text recognition, the optimal segmentation and recognition path is evaluated by the integrated model composed of segmentation, character recognition, linguistic context, and geometric context with the combining parameters optimized by the genetic algorithm (GA) [13] as shown in Eq. (5).

In relation to Eq. (5), UP is interpreted as either Sb or Sw in csr-lattice. When it is between character patterns, it is treated as Sb. When it is within a character pattern, it is treated as Sw. On the other hand, SP is always treated as Sb and NSP is always treated as Sw. The probabilities $P(g_j|Sb)$ and $P(g_j|Sw)$ are approximated by the SVM classifier.

### 3.2 Character Recognizer

Because Japanese has a large character set, two-level classification is used to reduce the time cost of recognition [15]. First, the coarse classification is performed to obtain candidates, and then the fine classification is executed.

#### 3.2.1 Coarse Classifier

First, we discard time-sequence information from an input on-line pattern. Second, we use the pseudo 2D bi-moment normalization (P2DBMN) [16] to absorb non-linear distortion of patterns to some extent. Third, we extract 8-directional features from $8 \times 8$ regions of an input pattern and blur them by the low-pass Gaussian filter so that we can extract a 512-directional feature vector [17]. To increase robustness against stroke connection, we extract two types of directional features: one from strokes and the other not only from strokes but also from off-strokes. Fourth, we transform each element of the feature vector by the Box-Cox transformation to improve the Gaussianity of feature distribution. Lastly, we use a simple distance measure, i.e., the Euclidian distance, to select candidates from 7,097 categories.

The directional features are more robust but less distinctive than time-sequence features, so they are suitable for coarse classification. Compression of the coarse classifier is described in Sect. 4.2.

#### 3.2.2 Fine Classifier

For the fine classification, we use an on-line recognizer and an off-line recognizer to enhance robustness and combine them by the sum rule.

The off-line recognizer uses the same non-linear normalization and feature extraction as the coarse classifier. The difference is the evaluation function. We use MQDF2, which is described in Sect. 3.2.4, instead of the Euclidian distance.

On the other hand, the on-line recognizer uses linear normalization to keep the horizontal and vertical ratio and extracts feature points to improve robustness and efficiency. For each stroke, first, the start and end points are picked up as feature points; then, the point farthest from the straight line through adjacent feature points is selected as a feature point while the distance is greater than a specified threshold. This process continues recursively until no more feature points are selected [18]. Then, it is applied to LTM as

mentioned in Sect. 1.

LTM is well combined with structured character pattern representation where a representation of an on-line pattern by a sequence of feature points is very compact and on-line patterns for radicals (sub-patterns) are shared among character patterns so that the on-line recognizer only occupies 252 KB for 7,097 character categories.

On the other hand, MRF is a statistical model, so sharing of radical patterns is not straightforward as a result of its larger memory size. Here, the details of LTM and MRF are not required for the rest of this paper.

#### 3.2.3 Classifier Combination

The two recognizers are combined as follows [19]:

A pattern $s = (s_1, \cdots, s_m)^T$ is recognized as a character class $c_i$ by the on-line and off-line recognizers with their evaluation scores $f_{on}^{c_i}$ and $f_{off}^{c_i}$, respectively. Then, the confidence of the combined recognizer $f_{off}^{c_i}$ by the sum rule with class-independent linear combining parameters is given by Eq. (7).

$$f_{com}^{c_i} = \lambda_1 f_{on}^{c_i} + \lambda_2 f_{off}^{c_i}. \tag{7}$$

We use the MCE criterion [20] to optimize the parameters. On each training pattern, a loss function is computed to approximate the classification error and the empirical loss is minimized to optimize the combining parameters on training patterns. The misclassification measure of a pattern from the class $c_i$ is given by Eq. (8), where $f_{com}^{r}$ is the evaluation of the closest rival class. The misclassification measure is transformed to give the loss as given by Eq. (9).

$$u_{c_i} = f_{com}^{r} - f_{com}^{c_i}, \tag{8}$$

$$ls_{c_i}(s) = ls_{c_i}(u_{c_i}) = \frac{1}{1 + \exp(-\xi u_{c_i})}. \tag{9}$$

On all the training patterns $\{(s^n, c^n) | n = 1, 2, \cdots, N\}$, the empirical loss is computed by Eq. (10) where $c^n$ is the class label of pattern $s^n$, $I(\cdot)$ is an indicator function and $M$ denotes the number of character classes. We minimize $f_L$ by stochastic gradient descent to obtain the optimal parameters.

$$f_L = \frac{1}{N} \times \sum_{n=1}^{N} \sum_{i=1}^{M} ls_i(s^n) \times I(c^n \in i). \tag{10}$$

Since the LTM on-line recognizer occupies just 252 KB but the MQDF2 off-line recognizer requires about 90 MB, compression of the off-line recognizer is the key task for building the compact combined recognizer, which is detailed in Sect. 5.2.

#### 3.2.4 MQDF2

MQDF2 is a smoothed version of the MQDF in which the minor eigenvalues are replaced with a larger constant. For an input character pattern $X = (x_1, \ldots, x_n)^T$, the MQDF2 for class $\omega_i$, $i = 1, 2, \ldots, N$, is obtained as follows:

$$g_2(X, w_i) = \sum_{j=1}^{k} \frac{1}{\lambda_{ij}} [\varphi_{ij}^T (X - u_i)]^2 + \frac{1}{\delta_i} D_c(X) + \sum_{j=1}^{k} \log \lambda_{ij} + (n - k) \log \lambda_{ij}, \tag{11}$$

where $\lambda_{ij}$ and $\varphi_{ij}$, $j = 1, 2, \ldots, k$, denote the eigenvalues of covariance of class $\omega_i$ sorted in decreasing order and the corresponding eigenvectors, respectively; $k$ denotes the number of principal components and $D_c(X)$ is the squared Euclidean distance in the complement subspace given by Eq. (12); the parameter $\delta_i$ can be set as a class-independent constant as proposed by Kimura et al. [4], and $tr(\sum_i)$ denotes the trace of covariance.

$$D_c(X) = \|x - u_i\|^2 - \sum_{j=1}^{k} \left[ \varphi_{ij}^T (x - u_i)^2 \right], \tag{12}$$

$$\delta_i = \frac{tr(\sum_i) - \sum_{j=1}^{k} \lambda_{ij}}{n - k} = \frac{1}{n - k} \sum_{j=k+1}^{n} \lambda_{ij}. \tag{13}$$

From the above description, the size of the off-line prototype dictionary $S$ is dependent on the data type of the parameters $u_i$, $\lambda_{ij}$, $\varphi_{ij}$ and $\delta_i$ whose sizes are noted as $T_u$, $T_\lambda$, $T_\varphi$ and $T_\delta$, respectively. In our system, $T_u$, $T_\lambda$, $T_\varphi$ are integers with 2 bytes; $T_\delta$ is a long integer with 4 bytes. We can have the total size of the prototype dictionary given by Eq. (14) where N denotes the number of the character categories.

$$S = N \times \{(n \times T_u + k \times T_\varphi) + k \times T_\lambda + T_\delta\}. \tag{14}$$

#### 3.2.5 FDA

To reduce the computation complexity and improve the robustness, Fisher discriminant analysis (FDA) is widely used to reduce the dimensionality of feature vectors. Suppose there are $C$ known character classes $\omega_1, \omega_2, \ldots, \omega_C$ and the $i$th class with $N_i$ training samples, in total $N$ training samples. $X = \{x_j^i\}$ $(j = 1, 2, \ldots C,\ i = 1, 2, \ldots N_j)$ is a set of samples with $n$-dimensions where $N_j$ is the sample number of $j$th class. The mean vector of each class and all classes can be expressed as $\bar{X}_j = (1/N_j) \sum_{i=1}^{N_j} x_j^i$ and $\bar{X} = (1/N) \sum_{j=1}^{C} \sum_{i=1}^{N_j} x_j^i$, respectively. Let the between-class matrix and within-class scatter matrix be defined as:

$$S_b = \sum_{j=1}^{C} N_j (\bar{X}_j - \bar{X})(\bar{X}_j - \bar{X})^T, \tag{15}$$

$$S_w = \sum_{j=1}^{C} \sum_{i=1}^{N_j} (x_j^i - \bar{x}_j)(x_j^i - \bar{x}_j)^T. \tag{16}$$

The process to obtain the transformation matrix is maximizing the following quotient, called the Fisher discriminant criterion [21].

$$w_{opt} = argmax_w \left| \frac{w^T S_b w}{w^T S_w w} \right| = [w_1, w_2, \ldots w_m]. \quad (17)$$

where $\{w_k | k = 1, 2, \ldots, m\}$ are $m$ $n$-dimensional eigenvectors of $S_w^{-1} S_b$ corresponding to the $m$ largest eigenvalues. $w_{opt}$ is a $n \times m$ matrix composed of the $m$ $n$-dimensions eigenvectors. By the transformation matrix $w_{opt}$, we can reduce the feature dimensions from $n$-dimensions to $m$-dimensions.

## 4. Pattern Databases and Benchmark

This section describes pattern databases to train and evaluate the text and character recognizers. Then it presents the implementation and performance of the full-scale text recognizer as the benchmark to evaluate the compressed recognizer.

### 4.1 Pattern Databases

To train and evaluate the text and character recognizers and their components shown in Fig. 3, we use the TUAT HANDS databases: Kuchibue, Nakayosi, and Tehon databases of on-line handwritten Japanese characters, and Kondate database of on-line handwritten text. The Kuchibue database contains on-line handwritten character patterns of 120 writers, 11,962 patterns per writer covering 3,356 character classes with $11{,}962 \times 120 = 1{,}435{,}440$ patterns in total. The Nakayosi database contains the samples of 163 writers, 10,403 patterns per writer covering 4,438 character classes with $10{,}403 \times 163 = 1{,}695{,}689$ patterns in total. Although the Tehon database is written by only one writer, it contains all the character categories covering JIS 1st and 2nd sets of Kanji with the correct stroke number and order. While Kuchibue and Nakayosi collected each character pattern written in each writing box, Kondate sampled character patterns written freely without any writing grids and even without guidelines by 100 writers.

To build the text recognizer covering 7,097 categories, Kuchibue and Nakayosi are not sufficient since they have far fewer than 7,097 covered character categories. Therefore, we use Tehon and distort each sample by distortion models [22] to generate 300 artificial patterns, which are merged with the Kuchibue, Nakayosi and Tehon databases. Consequently, the merged database contains 3,930,886 character patterns, covering 7,097 character classes.

### 4.2 Full-Scale Text Recognizer as a Benchmark

On the basis of our previous works [3], [14], [18], [23], the full-scale text recognizer with the same block diagram shown in Fig. 3 was developed as the benchmark.

We use MQDF2 for the off-line character recognizer and MRF for the on-line character recognizer. For MQDF2, we extract a 512-dimensional directional feature vector and reduce it to 96-dimensional one by LDA since it provides the best recognition rate.

The coarse classifier and the character recognizer in the full-scale text recognizer were trained and evaluated by the merged database, within which 4/5 of patterns of each class were used as training patterns noted as *TrPInM* and the remaining 1/5 of patterns were used as testing patterns noted as *TPInM*.

**Table 3** Performance of character recognizers.

| Character recognizer | Accuracy | Time cost/character |
|---|---|---|
| On-line recognizer | 91.16% | 27.5 ms |
| Off-line recognizer | 91.88% | 6.11 ms |
| Combined recognizer | 93.84% | 3.08 ms |

**Table 4** Memory size and accuracy of the full-scale text recognizer.

| Components | Memory size |
|---|---|
| SVM classifier for segmentation | 2.29 MB |
| Linguistic context processor | 10.5 MB |
| Geometric context evaluation module | 613 KB |
| Combined character recognizer | 50.5 MB |
| Total size | 64.1 MB |
| Accuracy | 93.6% |

The accuracy of the coarse classifier to nominate the correct class within the top 20 candidates is 99.91% by 5-fold cross-validation on the merged database.

Accuracies and time costs of the on-line, off-line, and combined recognizers by 5-fold cross-validation on the merged database are shown in Table 3.

Then, to train the combining parameters of each component in the full-scale text recognizer, we extracted horizontally written text lines from the Kondate database. The text lines written by 75 writers were used as the training patterns noted as *TrPInK* to train the SVM classifier and the combining parameters of each component in the text recognizer. The remaining text lines from the other 25 writers in Kondate were applied as testing patterns noted as *TPInK* to evaluate the text recognizer. Results showed that the accuracy reached 93.6% but the memory size needed to be about 64.1 MB.

The linguistic context model was trained by the 1993 volume of the Asahi newspaper and the 2002 volume of the Nikkei newspaper. After pruning low tri-gram occurrences, the linguistic context processor of 10.5 MB was obtained.

Table 4 shows the memory requirement for the components and accuracy of the full-scale text recognizer. Although the full-scale text recognizer has been used for on-the-shelf tablet PCs, this memory cost is the biggest obstacle to the recognizer's use in hand-held devices.

## 5. Construction of Compact Text Recognizer

This section describes construction of a compact text recognizer that is composed by compression of its components.

### 5.1 Compression of Coarse Classifier

For the Euclidean distance, the prototype of each class just needs to store the mean vector. There are two methods to compress its representation, i.e., vector quantization (VQ)

**Table 5** Comparison between data scaling and VQ for coarse classification.

| Compression method | Accuracy of Top 20 | Time cost per character |
|---|---|---|
| Data scaling | 99.7161% | 4.7977 (ms) |
| VQ | 99.7144% | 4.7955 (ms) |

or data scaling. We have compared both methods by *TPInM* but could not find notable differences as shown in Table 5.

A simple data scaling method transforms the value range of each element in the mean vector from a short integer to an unsigned char. Given $u_{max}$ and $u_{min}$ as the maximum and minimum value of all the elements of mean vectors $u_i = (u_i^1, \cdots, u_i^k)^T$ $(i = 1, \cdots, N)$, all the elements in the mean vectors can be scaled into a relative small range $T_R$ by Eq. (18). Typically, $T_R$ takes 255 in our compact system, because it can be stored by just unsigned char.

$$y_i^k = \frac{u_i^k - u_{min}}{u_{max} - u_{min}} \times T_R. \tag{18}$$

As described above, we extract two types of directional features: one from strokes and the other from strokes and off-strokes after non-linear normalization is applied. For both types, we extract $8 \times 8$ regions $\times$ 8 directions = 512 directional features and then reduce them into a 50-dimensinal vector by LDA.

Consequently, two prototype dictionaries covering 7,097 classes and two corresponding transformation matrixes used by LDA require a 1.56 MB memory in total.

### 5.2 Compression of MQDF2

From Eq. (14) in Sect. 3.2.4, we can see that the memory problem is mainly caused by eigenvectors $\varphi_{ij}$ $(i = 1, \cdots, N;\ j = 1, \cdots, k)$. Typically, the required memory size for all the eigenvectors $\varphi_{ij}$ $(i = 1, \cdots, N;\ j = 1, \cdots, k)$ reaches $N \times n \times k \times T_\varphi$ bytes. When $T_\varphi$ denotes the size of a short integer and $N$, $k$ and $n$ take 7,097, 160 and 40, respectively, eigenvectors of MQDF2 requires about 90 MB memory. Moreover, the mean vector $u_i$ $(i = 1, \cdots, N)$ is also scalable to compress, although it requires much less memory than eigenvectors $\varphi_{ij}$ $(i = 1, \cdots, N; j = 1, \cdots, k)$.

In the achievement of MQDF2, to reduce computation time cost while improving the accuracy, dimensionality of the feature vector is usually compressed from the original 512-dimensional to D-dimensional by LDA. By choosing the top 40 dominant components ($k = 40$), the parameters of eigenvectors are further compressed. Moreover, the latter elements in dominant eigenvectors are much smoother than the former elements, which ensures us to further reduce the elements of each dominant eigenvector to $D_L$ $(D_L < D)$ by substituting the latter $D - D_L$ elements by their mean value.

To compress dominant eigenvectors of each class further while keeping high accuracy, a kind of loss data compression based on the principle of block coding named vector quantization (VQ) [24] is used. First, split all the eigenvectors of each class into multiple sub-eigenvectors. Typically, each $D_L$-dimensional eigenvector $\varphi_{ij}$ $(i = 1, \cdots, N;\ j = 1, \cdots, k)$ is equally divided into $Q$ $D_Q$-dimensional sub-vectors $\varphi_{ij}^1, \varphi_{ij}^2, \cdots, \varphi_{ij}^Q$ with $D_L = D_Q \times Q$. Based on the method [6], we further use the LBG algorithm by minimum squared error (MSE) measure to cluster all the sub-vectors $\varphi_{ij}^q$ $(i = 1, \cdots, N;\ j = 1, \cdots, k; q = 1, \cdots, Q)$ into $L$ $(L \ll N \times k \times Q)$ clusters where cluster centers are noted as $p_l$ $(l = 1, \cdots, L)$, which are $D_Q$-dimensional. When $\varphi_{ij}^q$ $(i = 1, \cdots, N;\ j = 1, \cdots, k;\ q = 1, \cdots, Q)$ being classified to $m$-th cluster, it is substituted by its cluster center $p_m$ during the process of VQ. Therefore, sub-vectors in eigenvectors can be represented by indexes of corresponding cluster centers. Furthermore, we not only compress eigenvectors but also employ VQ to mean vectors $u_i$ $(i = 1, \cdots, N)$ of MQDF2. Because mean vectors include a relatively small number of elements, the configuration $D_Q = 1$ and $L = 256$ of vector quantization is used to mean vector compression.

After parameter compression of eigenvectors and mean vectors, the required memory for the compressed MQDF2 is calculated as follows: the memory for eigenvectors is composed of two parts: the indices of the prototypes and the codebook as shown by Eq. (19). The memory for mean vectors is shown by Eq. (20). Therefore, the memory for the compressed MQDF2 is given by Eq. (21).

$$size(\varphi) = \frac{\log_2^L}{8 \times D_Q} \times D_L \times N \times k + D_Q \times L \times T_\varphi, \tag{19}$$

$$size(u) = \frac{\log_2^L}{8 \times D_Q} \times D_L \times N + D_Q \times L \times T_u, \tag{20}$$

$$S_c = size(\varphi) + size(u) + N \times k \times T_\lambda + N \times T_\delta. \tag{21}$$

### 5.3 Compression of Linguistic Context

The linguistic context model for text recognizer is reduced from 10.5 MB to 5 MB and further to 2 MB by pruning low tri-gram occurrences in accordance with different thresholds.

### 5.4 Compression of Geometric Context

In the geometric context, $P(p_i^b | c_{i-1} c_i)$ requires large memory cost, since it is two dimensional. Due to the fact that many characters share the similar geometric shape and relation to adjacent characters, the character classes are clustered into six groups $s_k$ $(k = 1, \cdots, 6)$ by the LBG algorithm. Given $c_{i-1}$ and $c_i$ assigned to $s_1$ and $s_2$, respectively, $P(p_i^b | c_{i-1} c_i)$ is approximated by $P(p_i^b | s_1 s_2)$. Then, the memory of geometric context is reduced to about 610 KB.

### 5.5 Compression of SVM Classifier for Segmentation

The SVM classifier for segmentation point discrimination is trained by *TrPInK* in the same way as the full-scale text recognizer.

To compress it, however, we only use 11 elements of the geometric feature vector concerning segmentation point

**Table 6** Size of the text recognizer except the character recognizer.

| Components | Memory size |
|---|---|
| SVM classifier for segmentation | 319 KB |
| Linguistic context processor | 10.5 MB, 5 MB or 2 MB |
| Geometric context evaluation module | 610 KB |
| Total size | 11.5 MB, 6 MB or 3 MB |

$g_j$, namely, $f_2$, $f_6$, $f_9$, $f_{10}$, $f_{11}$, $f_{13}$, $f_{14}$, $f_{15}$, $f_{16}$, $f_{17}$, $f_{18}$ shown in Table 1, to reduce the memory cost from 2.3 MB to 319 KB.

### 5.6 Total Effect of the Above Compression

From the above compression, the memory sizes of the components and the total compact text recognizer except the character recognizer are listed in Table 6. Compression of the character recognizer is discussed in Sect. 6.

## 6. Experiments

Since the compressed MQDF2 character recognizer plays the most important role, we first describe experiments on it and its parameter selection, then compare different VQs, and finally present the evaluation on the compact text recognizer.

### 6.1 Evaluation of Compressed MQDF2

Table 7 shows the recognition accuracies and the corresponding memory by the different $D_L$ and $k$ where $D_L$ takes 60, 72 or 84 and $k$ takes 2 to 12. The last line in Table 7 (when $D_L$ is 96) shows the recognition accuracy and corresponding memory of the original MQDF2 recognizer described in Sect. 3.2.4. Compared with that, the memory is compressed from 17.2 MB to 11.4 MB when $D_L$ is 60 and $k = 12$ in Table 7 (b) with accuracy loss of about 0.7% from 91.87% to 91.18% in Table 7 (a). Mean vectors and eigenvectors are further compressed as described in Sect. 5.2. When $D_Q = 1$ and $L = 256$, the accuracy and corresponding memory size of the compressed MQDF2 recognizer on different parameter settings are listed in Table 8. From Tables 7 and 8, the memory size is at maximum decreased by over 1/2 (from 17.2 MB to 7.83 MB) with only 0.4% accuracy loss maximally after compression of MQDF2.

### 6.2 Comparison of Different VQs

From comparing in Tables 7 and 8, we can learn that the memory size is apparently reduced with just a little accuracy loss after compression to MQDF2. Moreover, the parameters $D_Q$ and $L$ of VQ may bring about many different configurations in combination with $D_L$, so that these different parameter configurations need to be compared to obtain the optimal configuration. The figure about tradeoff between accuracy and memory size (accuracy-size tradeoff) of the compressed MQDF2 with different $D_L$, $D_Q$ and $L$ configurations is used to find which parameter configuration is optimal. The smaller the number of cluster centers $L$, the bigger the distortion error, so we just take 256 for $L$, which can be stored by unsigned char.

**Table 7** (a) Recognition accuracy (%) and (b) corresponding memory size (MB) on different $k$ and $D_L$.

(a) Accuracies (%)

| $D_L$ | Dominant eigenvector number $k$ | | | | |
|---|---|---|---|---|---|
| | 2 | 5 | 8 | 10 | 12 |
| 60 | 89.82 | 90.68 | 90.98 | 91.10 | 91.15 |
| 72 | 89.84 | 90.91 | 91.29 | 91.44 | 91.53 |
| 84 | 89.82 | 91.02 | 91.45 | 91.67 | 91.77 |
| 96 | 89.75 | 91.03 | 91.52 | 91.78 | 91.92 |

(b) Memory cost (MB)

| $D_L$ | Dominant eigenvector number $k$ | | | | |
|---|---|---|---|---|---|
| | 2 | 5 | 8 | 10 | 12 |
| 60 | 3.01 | 5.52 | 8.04 | 9.72 | 11.4 |
| 72 | 3.33 | 6.34 | 9.34 | 11.3 | 13.3 |
| 84 | 3.66 | 7.15 | 10.6 | 12.9 | 15.3 |
| 96 | 3.97 | 7.95 | 11.9 | 14.5 | 17.2 |

**Table 8** (a) Recognition accuracy (%) and (b) corresponding memory size (MB) on different $k$ and $D_L$ when $D_Q = 1$ and $L = 256$.

(a) Accuracies (%)

| $D_L$ | Dominant eigenvector number $k$ | | | | |
|---|---|---|---|---|---|
| | 2 | 5 | 8 | 10 | 12 |
| 60 | 89.82 | 90.58 | 90.91 | 91.08 | 91.01 |
| 72 | 89.78 | 90.84 | 91.15 | 91.31 | 91.29 |
| 84 | 89.80 | 90.88 | 91.45 | 91.55 | 91.51 |

(b) Memory cost (MB)

| $D_L$ | Dominant eigenvector number $k$ | | | | |
|---|---|---|---|---|---|
| | 2 | 5 | 8 | 10 | 12 |
| 60 | 1.59 | 2.84 | 4.14 | 5.01 | 5.88 |
| 72 | 1.71 | 3.25 | 4.79 | 5.82 | 6.85 |
| 84 | 1.87 | 3.66 | 5.44 | 6.63 | 7.83 |

To find the optimal configuration of $D_L$ and $D_Q$, experiments were performed on different $D_L$ and $D_Q$. Table 9 shows the recognition accuracy and corresponding memory size for their combinations. To find which $D_Q$ is best, Figure 5 shows the accuracy-size tradeoff for the compressed MQDF2 recognizer on different $D_Q$ while $D_L = 72$. From Fig. 5, we can learn that $D_Q = 2$ is better than others, because not only its memory size is small at about 3 MB but also the accuracy loss is very small at only about 0.86% compared with the original MQDF2 in Table 7. Similarly, to determine which $D_L$ is best, the accuracy-size tradeoff for the compressed recognizer on different $D_L$ with $D_Q = 2$ is shown in Fig. 6, and $D_L = 72$ is found to be better than others.

### 6.3 Evaluation of Compact Text Recognizer

Since the off-line character recognizer is the key to compression, we just use the parameter configuration $D_L$, $k$ and $D_Q$ of the compressed off-line recognizer to label the compact text recognizer. *TPInK* is used to evaluate the compact text recognizer. Table 10 shows text recognition accuracy for different parameter configuration: $D_L$, $k$ and $D_Q$.

Although we have determined that the configuration $D_L = 72$ and $D_Q = 2$ for the compressed off-line recognizer

**Table 9** Recognition accuracy (%) and corresponding memory size (MB) on different $k$ and $D_L$.

(a) When $D_Q = 2$ and $L = 256$

Accuracy (%)

| $D_L$ | Dominant eigenvector number $k$ | | | | |
|---|---|---|---|---|---|
| | 2 | 5 | 8 | 10 | 12 |
| 60 | 89.58 | 90.41 | 90.51 | 90.83 | 90.86 |
| 72 | 89.66 | 90.66 | 90.86 | 91.02 | 91.01 |
| 84 | 89.67 | 90.67 | 90.93 | 91.26 | 91.20 |

Memory cost (MB)

| $D_L$ | Dominant eigenvector number $k$ | | | | |
|---|---|---|---|---|---|
| | 2 | 5 | 8 | 10 | 12 |
| 60 | 1.14 | 1.83 | 2.52 | 2.98 | 3.44 |
| 72 | 1.22 | 2.03 | 2.84 | 3.39 | 3.93 |
| 84 | 1.30 | 2.24 | 3.17 | 3.80 | 4.41 |

(b) When $D_Q = 3$ and $L = 256$

Accuracy (%)

| $D_L$ | Dominant eigenvector number $k$ | | | | |
|---|---|---|---|---|---|
| | 2 | 5 | 8 | 10 | 12 |
| 60 | 89.73 | 90.21 | 90.09 | 90.29 | 90.26 |
| 72 | 89.75 | 90.38 | 90.42 | 90.46 | 90.68 |
| 84 | 89.38 | 90.52 | 90.68 | 90.66 | 90.79 |

Memory cost (MB)

| $D_L$ | Dominant eigenvector number $k$ | | | | |
|---|---|---|---|---|---|
| | 2 | 5 | 8 | 10 | 12 |
| 60 | 1.00 | 1.49 | 1.98 | 2.30 | 2.63 |
| 72 | 1.06 | 1.63 | 2.20 | 2.57 | 2.96 |
| 84 | 1.11 | 1.76 | 2.41 | 2.85 | 3.28 |

(c) When $D_Q = 4$ and $L = 256$

Accuracy (%)

| $D_L$ | Dominant eigenvector number $k$ | | | | |
|---|---|---|---|---|---|
| | 2 | 5 | 8 | 10 | 12 |
| 60 | 89.39 | 89.78 | 90.19 | 89.93 | 89.92 |
| 72 | 89.45 | 90.11 | 90.08 | 90.18 | 90.14 |
| 84 | 89.58 | 90.05 | 90.31 | 90.36 | 90.39 |

Memory cost (MB)

| $D_L$ | Dominant eigenvector number $k$ | | | | |
|---|---|---|---|---|---|
| | 2 | 5 | 8 | 10 | 12 |
| 60 | 0.94 | 1.32 | 1.71 | 1.96 | 2.22 |
| 72 | 0.98 | 1.42 | 1.87 | 2.17 | 2.46 |
| 84 | 1.02 | 1.52 | 2.03 | 2.37 | 2.71 |

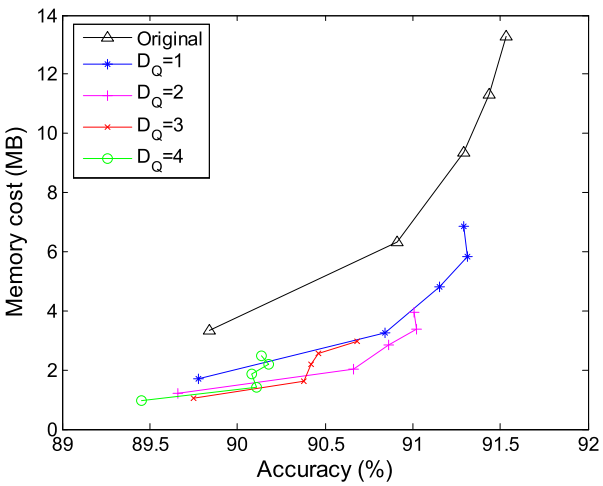



**Fig. 5** Tradeoff of the compressed MQDF2 on different $D_Q$ with $D_L = 72$.

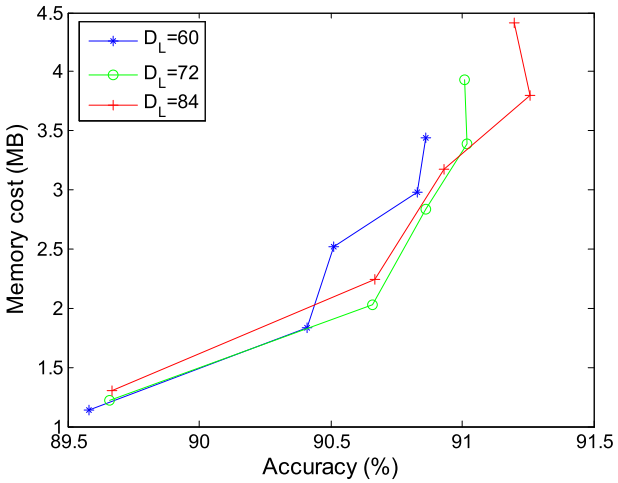



**Fig. 6** Tradeoff of the compressed MQDF2 on different $D_L$ with $D_Q = 2$.

**Table 10** Text recognition accuracy (%) with the compressed off-line recognizer on different $k$ and $D_L$.

(a) When $D_Q = 1$ and $L = 256$

| $D_L$ | Dominant eigenvector number $k$ | | | | |
|---|---|---|---|---|---|
| | 2 | 5 | 8 | 10 | 12 |
| 60 | 92.84 | 93.06 | 93.03 | 93.04 | 93.05 |
| 72 | 92.97 | 93.21 | 93.17 | 93.25 | 93.26 |
| 84 | 93.03 | 93.44 | 93.34 | 93.37 | 93.40 |

(b) When $D_Q = 2$ and $L = 256$

| $D_L$ | Dominant eigenvector number $k$ | | | | |
|---|---|---|---|---|---|
| | 2 | 5 | 8 | 10 | 12 |
| 60 | 92.85 | 92.97 | 92.97 | 93.05 | 93.06 |
| 72 | 92.85 | 93.26 | 93.13 | 93.17 | 93.18 |
| 84 | 92.99 | 93.38 | 93.26 | 93.35 | 93.34 |

(c) When $D_Q = 3$ and $L = 256$

| $D_L$ | Dominant eigenvector number $k$ | | | | |
|---|---|---|---|---|---|
| | 2 | 5 | 8 | 10 | 12 |
| 60 | 92.74 | 92.98 | 92.95 | 92.93 | 93.01 |
| 72 | 92.93 | 93.11 | 93.16 | 93.10 | 93.26 |
| 84 | 92.92 | 93.34 | 93.19 | 93.27 | 93.32 |

(d) When $D_Q = 4$ and $L = 256$

| $D_L$ | Dominant eigenvector number $k$ | | | | |
|---|---|---|---|---|---|
| | 2 | 5 | 8 | 10 | 12 |
| 60 | 92.73 | 92.81 | 92.85 | 92.75 | 92.73 |
| 72 | 92.69 | 92.97 | 93.00 | 93.09 | 93.04 |
| 84 | 92.80 | 93.10 | 93.24 | 93.08 | 93.11 |

is better than other configurations for character recognition, $D_L$ may be reduced and $D_Q$ may be increased further because the linguistic and geometric context both push character recognition in the right direction in text recognition. For this reason, we consider 60 besides 72 for $D_L$ and 3 and 4 besides 2 for $D_Q$ during the process of drawing tradeoff between memory size of compressed off-line recognizer and accuracy of text recognition. If we found 60 were better than 72 for $D_L$, we would need to test even smaller values for $D_L$. Similarly, if we found 4 were better than 2 and 3 for $D_Q$, we would need to test even larger values for $D_Q$.

Figure 7 plots the tradeoff between the memory size of the compressed off-line recognizer and the accuracy of text

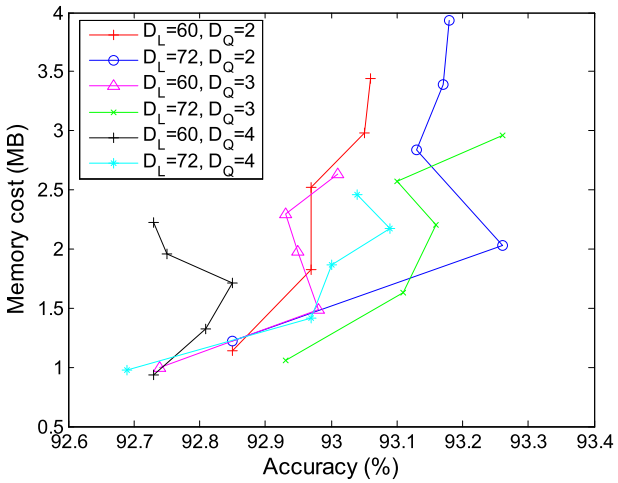



**Fig. 7** Tradeoff between memory size (MB) of the compressed off-line recognizer and text recognition accuracy (%).

recognition on different $D_Q$ and $D_L$.

Figure 7 shows that parameter configuration $D_L = 72$ and $D_Q = 3$ of the compressed off-line recognizer is better than other configurations for text recognition. This implies that we do not need to test values smaller than 60 for $D_L$ or larger than 4 for $D_Q$. Therefore, the compressed off-line recognizer where $D_L = 72$, $D_Q = 3$ and $k = 5$ with the 1.67 MB memory shown in Table 10 (b) is selected to build the compressed combined character recognizer. In this case, the compressed combined character recognizer used for text recognition requires about 3.5 MB, composed of the coarse classifier of 1.56 MB, the LTM on-line recognizer of 252 KB, and the selected compressed off-line recognizer of 1.67 MB. Consequently, the compact text recognizer requires only 14.9 MB, composed of the SVM classifier of 319 KB, the linguistic context of 10.5 MB and the geometric context of 610 KB, all together 11.5 MB as described in Table 6 and the selected compressed combined character recognizer of 3.5 MB. Although its size has been drastically reduced, it keeps the accuracy of 93.11%. We call it the 15-MB text recognizer. Compared with the full-scale text recognizer described in Sect. 4.2, the memory is reduced from 64.1 to 14.9 MB with only 0.5% accuracy loss.

By selecting the linguistic context of 5 MB while using a slightly larger configuration (2.19 MB) for the compressed MQDF2 with $D_L = 80$, $D_Q = 2$ and $k = 5$, with other components unchanged, a much smaller compact text recognizer with only 10.7 MB memory cost is obtained while keeping the accuracy of 92.8%, which is named the 11-MB text recognizer.

Finally, similar to above, we select the linguistic context of 2 MB and the compressed MQDF2 with $D_L = 60$, $D_Q = 4$ and $k = 2$ whose size is 963 KB to construct an even smaller compact text recognizer. The memory cost and the accuracy are 5.6 MB and 90.02%, respectively. We call it the 6-MB text recognizer.

From the above, Table 11 lists the accuracy and memory costs of compact text recognizers.

**Table 11** Accuracy and memory cost of compact text recognizer.

| Text recognition, size and accuracy | 15-MB recognizer | 11-MB recognizer | 6-MB recognizer |
|---|---|---|---|
| SVM classifier for segmentation | 319 KB | 319 KB | 319 KB |
| Linguistic context processor | 10.5 MB | 5 MB | 2 MB |
| Geometric context evaluation module | 610 KB | 610 KB | 610 KB |
| Coarse classifier | 1.56 MB | 1.56 MB | 1.56 MB |
| On-line recognizer | 252 KB | 252 KB | 252 KB |
| Off-line recognizer | 1.67 MB | 2.19 MB | 963 KB |
| Total memory cost | 14.9 MB | 10.7 MB | 5.6 MB |
| Accuracy | 93.11% | 92.80% | 90.02% |

**Table 12** Accuracy of text recognizer with on-line or off-line recognizer alone for character recognition in text recognition.

| On-line character recognition alone for text recognition | | Off-line character recognition alone for text recognition | |
|---|---|---|---|
| Text recognizer | Accuracy | Text recognizer | Accuracy |
| 15-MB text recognizer (13.2 MB) | 90.81% | 15-MB text recognizer (14.7 MB) | 92.97% |
| 11-MB text recognizer (9.1 MB) | 90.44% | 11-MB text recognizer (10.5 MB) | 92.62% |
| 6-MB text recognizer (4 MB) | 89.01% | 6-MB text recognizer (5.4 MB) | 89.71% |

**Table 13** Comparison in time cost of the compact text recognizers with full-scale text recognizer.

| Text recognition system | Time cost per line |
|---|---|
| Full-scale text recognizer | 0.615 s |
| 11-MB text recognizer | 0.666 s |
| 15-MB text recognizer | 0.684 s |
| 6-MB text recognizer | 0.665 s |

In comparison to the combined character recognizer, Table 12 shows the accuracy when either of the on-line recognizer or the off-line recognizer is employed alone instead of the combined character recognizer for character recognition in above described text recognizers of 6-MB, 11-MB and 15-MB. Since the other recognizer is not combined, the actual size is shown in parentheses.

To evaluate the speeds of 6-MB, 11-MB, 15-MB and a full-scale text recognizer, we experimented on a notebook PC with Intel Core 2 (1.33 GHz, 784 MHz) and 1 GB RAM. Table 13 lists the time costs of different text recognizers where the testing patterns in Kondate were used. We can see the time costs are almost the same.

### 6.4 Analysis and Discussion

From the above experiments, $D_Q = 2$ is optimal in the MQDF2 compression for character recognition described in Sect. 6.2, but $D_Q = 3$ is the best for text recognition described in Sect. 6.3. This is understandable, since the linguistic context processor and geometric context evaluation module in the text recognizer cooperate with the character recognizer.

From Table 13, the time cost per text line in the com-

pact text recognizers increases by 0.07 s compared with the full-scale text recognizer, but it is very tiny and can be accepted since each line contains 16.9 characters on average.

## 7. Conclusion

We have developed a robust and compact on-line handwritten Japanese text recognizer. To reduce the memory size of the recognizer, we used both intrinsic and technical methods to each component of the text recognizer and overall architecture. While the reduction of the memory is significantly large, that in recognition accuracy is kept remarkably small. i.e., memory reduced from 64.1 MB to 14.9 MB while accuracy reduced 0.5% from the best accuracy, 93.6%. The developed recognizer has been bundled into on-the-shelf smart phone. The approach to building a compact handwritten Japanese text recognizer can be extended to other large character set languages such as Chinese.

### References

[1] H. Tanaka, K. Nakajima, K. Ishigaki, K. Akiyama, and M. Nakagawa, "Hybrid pen-input character recognition system based on integration of online-offline recognition," Proc. ICDAR 99, pp.209–212, 1999.
[2] X.-D. Zhou and C.-L. Liu, "Online handwritten Japanese character string recognition incorporating geometric context," Proc. 9th ICDAR, pp.48–52, Curitiba, Brazil, 2007.
[3] M. Nakagawa, B. Zhu, and M. Onuma, "A model of on-line handwritten Japanese text recognition free from line direction and writing format constraints," IEICE Trans. Inf. & Syst., vol.E88-D, no.8, pp.1815–1822, Aug. 2005.
[4] F. Kimura, K. Takashina, S. Tsuruoka, and Y. Miyake, "Modified quadratic discriminant functions and the application to Chinese character recognition," IEEE Trans. Pattern Anal. Mach. Intell., vol.PAMI-9, no.1, pp.149–153, 1987.
[5] C.-L. Liu, R. Mine, and M. Koga, "Building compact classifier for large character set recognition using discriminative feature extraction," Proc. 8th ICDAR, pp.846–850, Seoul, Korea, 2005.
[6] T. Long and L. Jin, "Building compact MQDF classifier for large character set recognition by subspace distribution sharing," Pattern Recognit., vol.41, pp.2916–2925, 2008.
[7] K. Yoshida and H. Sakoe, "Online handwritten character recognition for a personal computer system," IEEE Trans. Consum. Electron., vol.CE-28, no.3, pp.202–209, 1982.
[8] K.H. Wong and F. Fallside, "Dynamic programming in the recognition of connected handwritten script," Proc. 2nd Artificial Intell. Appl., pp.666–670, 1985.
[9] M. Nakagawa and K. Akiyama, "A Linear-time elastic matching for stroke number free recognition of on-line handwritten characters," Proc. 4th IWFHR, pp.48–56, 1994.
[10] J. Hu and M.K. Brown, "HMM based online handwriting recognition," IEEE Trans. Pattern Anal. Mach. Intell., vol.18, no.10, pp.1039–1045, 1996.
[11] B. Zhu and M. Nakagawa, "On-line handwritten Japanese characters recognition using a MRF model with parameter optimization by CRF," Proc. ICDAR 2011, pp.603–609, Beijing, China, 2011.
[12] J. Gao, B. Zhu, and M. Nakagawa, "Complexity reduction with recognition rate maintained for on-line handwritten Japanese text recognition," Proc. SPIE, vol.8297, pp.1–8, Burlingame, USA, 2012.
[13] B. Zhu, X.-D. Zhou, C.-L. Liu, and M. Nakagawa, "A robust model for on-line handwritten Japanese text recognition," Int. J. Document Analysis and Recognition, vol.13, pp.121–131, 2010.
[14] B. Zhu and M. Nakagawa, "Segmentation of on-line freely written Japanese text using SVM for improving text recognition," IEICE Trans. Inf. & Syst., vol.E91-D, no.1 pp.105–113, Jan. 2008.
[15] B. Zhu and M. Nakagawa, "A coarse classifier construction method from a large number of basic recognizers for on-line recognition of handwritten Japanese characters," Proc. ICDAR 2011, pp.1090–1094, Beijing, China, 2011.
[16] C.-L. Liu and K. Marukawa, "Pseudo two-dimensional shape normalization methods for handwritten Chinese character recognition," Pattern Recognit., vol.38, pp.2242–2255, 2005.
[17] C.-L. Liu and X.-D. Zhou, "Online Japanese character recognition using trajectory-based normalization and direction feature extraction," Proc. 10th IWFHR, pp.217–222, 2006.
[18] U. Ramer, "An iterative procedure for the polygonal approximation of plan closed curves," Comput. Graph. Image Process., vol.1, pp.244–256, 1972.
[19] B. Zhu, J. Gao, and M. Nakagawa, "Objective function design for MCE-based combination of on-line and off-line character recognizers for on-line handwritten Japanese text recognition," Proc. ICDAR 2011, pp.594–599, Beijing, China, 2011.
[20] B.-H. Juang, W. Chou, and C.-H. Lee, "Minimum classification error rate methods for speech recognition," IEEE Trans. Speech Audio Process., vol.5, no.3, pp.257–265, 1997.
[21] M. Loog, R.P.W. Duin, and R. Haeb-Umbach, "Multiclass linear dimension reduction by weighted pairwise Fisher criteria," IEEE Trans. Pattern Anal. Mach. Intell., vol.23, no.7, pp.762–766, 2001.
[22] B. Chen, B. Zhu, and M. Nakagawa, "Effects of generating a large amount of artificial patterns for on-line handwritten Japanese character recognition," Proc. ICDAR 2011, pp.663–667, Beijing, China, 2011.
[23] M. Nakagawa, J. Tokuno, B. Zhu, M. Onuma, H. Oda, and A. Kitadai, "Recent results of online Japanese handwriting recognition and its applications," Arabic and Chinese Handwriting Recognition (D. Doermann and S. Jaeger ed. selected papers from SACH 2006), Lect. Notes Comput. Sci. (LNCS) 4768, pp.170–195, 2008.
[24] K.-W. Law and C.-F. Chan, "Split-dimension vector quantization of Parcor coefficients for low bit rate speech coding," IEEE Trans. Speech Audio Process., vol.2, no.3, pp.443–446, 1994.

**Jinfeng Gao** received the B.Sc. degree from University of Huanghuai, Zhumadian city, China and M.E. degree in Computer Science and Technology from Guizhou University, Guiyang, China, in 2004 and 2009, respectively. He is currently pursuing a Ph.D. degree in Engineering from Tokyo University of Agriculture and Technology (TUAT), Japan. His research interests include handwritten character recognition and text recognition.

**Bilan Zhu** received M.Sc. degree from Tokyo University of Agriculture and Technology (TUAT), Japan in 2004. In 2007, she received Ph.D. degree in Engineering from TUAT. She is now an Assistant Professor at TUAT. She is working on on-line, off-line handwriting recognition, documents processing and context analysis of Japanese, Chinese and English text.

**Masaki Nakagawa** was born on 31 October 1954 in Japan. He received B.Sc. and M.Sc. degrees from the University of Tokyo in 1977 and 1979, respectively. During the academic year 1977/78, he followed Computer Science course at Essex University in England, and received M.Sc. with distinction in Computer Studies in July 1979. He received Ph.D. in Information Science from the University of Tokyo in December 1988. From April 1979, he has been working at Tokyo University of Agriculture and Technology. Currently, he is a Professor of Media Interaction in Department of Computer and Information Sciences.