Design Compact Recognizers of Handwritten Chinese Characters Using Precision Constrained Gaussian Models, Minimum Classification Error Training and Parameter Compression [*]

Yongqiang Wang[1,2] and Qiang Huo[1]
[1]Microsoft Research Asia, Beijing, China
[2]Department of Computer Science, The University of Hong Kong, Hong Kong, China
(E-mails: yqwang@cs.hku.hk, qianghuo@microsoft.com)

Abstract

In our previous work, a precision constrained Gaussian model (PCGM) was proposed for character modeling to design compact recognizers of handwritten Chinese characters. A maximum likelihood training procedure was developed to estimate model parameters from training data. In this paper, we extend the above work by using minimum classification error (MCE) training to improve recognition accuracy and split vector quantization technique to compress model parameters. Compared with the state-of-the-art MCE-trained and compressed classifiers based on modified quadratic discriminant function, PCGM-based classifiers can achieve much better memory-accuracy tradeoff, therefore offer a good solution to designing compact handwriting recognition systems for East Asian languages such as Chinese, Japanese, and Korean.

1. Introduction

A state-of-the-art handwriting recognition system for East Asian (EA) languages such as Chinese, Japanese, and Korean typically includes a character classifier constructed by using a so-called modified quadratic discriminant function (MQDF) [10]. Researchers have been exploring different ways of reducing the memory requirement of MQDF-based classifiers, hoping to deploy it in mobile devices with limited memory (e.g., less than 2MB). A recent interesting experimental study was reported in [13], which was a straight-froward application of the relevant model compression techniques originally described in [8, 5, 6]. It was observed and confirmed by our own study that a tradeoff on recognition accuracy has to be made in order to construct a compact MQDF-based classifier.

In the past several years, we've being exploring other possible solutions for designing high-performance compact handwriting recognizers based on the concept of *structured covariance modeling*. So far, we have studied two modeling techniques, namely semi-tied covariance (STC) Gaussian model [16] and precision constrained Gaussian model (PCGM) [17, 18], respectively. In [16], detailed procedures are described for estimating the STC model parameters under both maximum likelihood (ML) and minimum classification error (MCE) criteria. In [17, 18], only ML training is studied to estimate PCGM parameters from large amount of training data. Without using model parameter compression techniques, both STC-based and PCGM-based classifiers can achieve a better memory-accuracy tradeoff than MQDF-based classifiers, yet PCGM-based approach seems more promising. In this study, we extend the above PCGM work by using MCE training to improve recognition accuracy and split vector quantization (VQ) [7] technique to compress model parameters, and compare its performance with that of the MCE-trained and compressed classifiers based on MQDF. It is the purpose of this paper to report the results and findings of this study.

The rest of this paper is organized as follows. After a brief description of MQDF and PCGM based approaches in section 2, we present our MCE-training and model compression procedures in section 3 and 4, respectively. This is followed by experimental results in section 5. Finally we conclude the paper in section 6.

2. MQDF and PCGM Based Approaches

Given a handwriting sample, we first extract a D_0-dimensional raw feature vector z using the procedures described in [1, 2, 3]. z will then be transformed into a lower dimensional feature space using a $D_0 \times D$ transformation matrix W, i.e., $x = W^T z$, where W is obtained by linear discriminant analysis (LDA) (e.g., [8]).

[*]This work has been done when the first author was an intern at Microsoft Research Asia, Beijing, China.

978-0-7695-3725-2/09 $25.00 © 2009 IEEE
DOI 10.1109/ICDAR.2009.41

Let $\{C_j | j = 1, \cdots, M\}$ be the set of M character classes, and $\mathcal{X} = \{(x_r, i_r) | r = 1, \cdots, R\}$ the set of training feature vectors, where x_r is the r-th training sample and i_r denotes the index of its true class label. Assume that feature vectors from the same character class C_j can be modeled by a Gaussian distribution with a mean vector μ_j and a full covariance matrix Σ_j. By setting the $(K+1)$-th to D-th eigenvalues of Σ_j as a class-dependent constant δ_j, a so-called MQDF can be defined as follows [10]:

$$g_j(x; \Theta_j) \triangleq -\frac{1}{2}\{\sum_{k=1}^{K} \log \rho_{jk} + (\frac{1}{\rho_{jk}} - \frac{1}{\delta_j}) p_{jk}^2 + (D-K)\log \delta_j + \frac{1}{\delta_j}\|x - \mu_j\|^2\} \quad (1)$$

where $\Theta_j = \{\mu_j, \delta_j, \rho_{jk}, v_{jk} | k = 1, \cdots, K\}$, ρ_{jk} and v_{jk} are the k-th leading eigenvalue and the corresponding eigenvector of Σ_j, $p_{jk} = (x - \mu_j)^T v_{jk}$, k and δ_j are two control parameters. In practice, setting δ_j as the average of $(K+1)$-th to D-th eigenvalues works well.

Under the same Gaussian assumption, PCGM [18] imposes a different constraint on each covariance matrix Σ_j. More specifically, each inverse covariance matrix $P_j = \Sigma_j^{-1}$ (a.k.a., precision matrix) is constrained to lie in a subspace spanned by a set of L basis symmetric matrices (referred to as "prototypes" hereinafter), $\Psi = \{S_l | l = 1, \cdots, L\}$, which are shared by all the character classes. Consequently, the precision matrix P_j can be written as $P_j \triangleq \sum_{l=1}^{L} \lambda_{jl} S_l$, where λ_{jl}'s are class-dependent basis coefficients (referred to as "coefficients" hereinafter) and L is a control parameter. The discriminant function of PCGM can be derived from log-likelihood function as

$$g_j(x; \mathbf{\Omega}) \triangleq \frac{1}{2}\left[\log\det(\sum_{l=1}^{L} \lambda_{jl} S_l) - \sum_{l=1}^{L} \lambda_{jl}(x - \mu_j)^T S_l (x - \mu_j)\right] \quad (2)$$

where $\mathbf{\Omega} = \{\mu_j, \lambda_{jl} | j = 1, \cdots, M; l = 1, \cdots, L\} \cup \Psi$. In practice, the above discriminant function can be evaluated more efficiently as follows:

$$g_j(x; \mathbf{\Omega}) \triangleq \frac{1}{2}\left(c_j + 2x^T m_j - \sum_{l=1}^{L} \lambda_{jl}\, x^T S_l x\right) \quad (3)$$

where $c_j = \log\det P_j - \mu_j^T P_j \mu_j$, and $m_j = P_j \mu_j$.

In recognition stage, an unknown feature vector x will be classified as the class with the maximum discriminant function value as follows:

$$x \in C_j \ \text{ if } \ j = \arg\max_{w} g_w(x) \ , \quad (4)$$

where $g_w(x)$ is the discriminant function of MQDF or PCGM, depending on which type of recognizer is used.

3. MCE Training of PCGMs Using Quickprop

In [18], we have described ML training procedure for PCGM. In this section, we present an MCE training procedure to further improve the recognition accuracy of PCGM-based classifiers, which is a straight-forward application of the general MCE formulation described in e.g., [9].

Given the discriminant function of PCGM in Eq. (2) and the decision rule in Eq. (4), a misclassification measure for each training sample x_r is defined first:

$$d_r(x_r; \mathbf{\Omega}) = -g_{i_r}(x; \mathbf{\Omega}) + G_{i_r}(x; \mathbf{\Omega}) \, , \quad (5)$$

where

$$G_j(x; \mathbf{\Omega}) = \frac{1}{\eta}\log\left[\frac{1}{M-1}\sum_{n, n\neq j}\exp[\eta g_n(x; \mathbf{\Omega})]\right] \quad (6)$$

with η being a control parameter. The PCGM parameters $\mathbf{\Omega}$ can then be estimated by minimizing the following empirical loss function $\ell(\mathbf{\Omega}; \mathcal{X})$

$$\ell(\mathbf{\Omega}; \mathcal{X}) = \frac{1}{R}\sum_{r=1}^{R}\frac{1}{1 + \exp(-\alpha d_r + \beta)} \quad (7)$$

where α and β are two control parameters.

Among several options, we use Quickprop algorithm [4] to minimize the objective function in Eq. (7) because of its effectiveness for MCE training as demonstrated in other pattern recognition applications (e.g., [14]). Starting from ML-trained seed PCGM parameters, the following iterative Quickprop procedure adapted from [14] is used to fine-tune the mean vectors, $\{\mu_j\}$, only:

Step 1: Let $t = 1$. Calculate the derivative of $\ell(\mathbf{\Omega}; \mathcal{X})$ w.r.t. each μ_{jd} and update it by

$$\mu_{jd}^{(t+1)} = \mu_{jd}^{(t)} - \varepsilon_0 \frac{\partial \ell(\mathbf{\Omega}^{(t)}; \mathcal{X})}{\partial \mu_{jd}}$$

where μ_{jd} is the d-th element of μ_j, $\frac{\partial \ell(\mathbf{\Omega}^{(t)}; \mathcal{X})}{\partial \mu_{jd}} \triangleq \frac{\partial \ell(\mathbf{\Omega}; \mathcal{X})}{\partial \mu_{jd}}|_{\mathbf{\Omega} = \mathbf{\Omega}^{(t)}}$, and ε_0 is an initial learning rate set empirically.

Step 2: Let $t \leftarrow t + 1$. Calculate the approximate second derivative of $\ell(\mathbf{\Omega}; \mathcal{X})$ w.r.t. each μ_{jd} as follows:

$$\frac{\partial^2 \ell(\mathbf{\Omega}^{(t)}; \mathcal{X})}{\partial \mu_{jd}^2} \approx \frac{\frac{\partial \ell(\mathbf{\Omega}^{(t)}; \mathcal{X})}{\partial \mu_{jd}} - \frac{\partial \ell(\mathbf{\Omega}^{(t-1)}; \mathcal{X})}{\partial \mu_{jd}}}{\mu_{jd}^{(t)} - \mu_{jd}^{(t-1)}} \, .$$

Step 3: Calculate update step differently depending on the following cases:

- If $\frac{\partial^2 \ell(\mathbf{\Omega}^{(t)};\mathcal{X})}{\partial \mu_{jd}^2} > 0$ and the sign of gradient $\frac{\partial \ell(\mathbf{\Omega}^{(t)};\mathcal{X})}{\partial \mu_{jd}}$ differs from that of $\frac{\partial \ell(\mathbf{\Omega}^{(t-1)};\mathcal{X})}{\partial \mu_{jd}}$, then the following Newton step is used:

$$\delta_t \mu_{jd} = -\frac{\partial \ell(\mathbf{\Omega}^{(t)};\mathcal{X})}{\partial \mu_{jd}} \Big/ \frac{\partial^2 \ell(\mathbf{\Omega}^{(t)};\mathcal{X})}{\partial \mu_{jd}^2} \,,$$

where $\delta_t \mu_{jd}$ denotes the update step of μ_{jd}.

- If $\frac{\partial^2 \ell(\mathbf{\Omega}^{(t)};\mathcal{X})}{\partial \mu_{jd}^2} > 0$ and $\frac{\partial \ell(\mathbf{\Omega}^{(t)};\mathcal{X})}{\partial \mu_{jd}}$ and $\frac{\partial \ell(\mathbf{\Omega}^{(t-1)};\mathcal{X})}{\partial \mu_{jd}}$ have the same sign, the following modified Newton step is used:

$$\delta_t \mu_{jd} = -\left(1 \Big/ \frac{\partial^2 \ell(\mathbf{\Omega}^{(t)};\mathcal{X})}{\partial \mu_{jd}^2} + \varepsilon_t\right) \frac{\partial \ell(\mathbf{\Omega}^{(t)};\mathcal{X})}{\partial \mu_{jd}} \,,$$

with ε_t being a learning rate set as follows:

$$\varepsilon_t = \varepsilon_0 (1 - t/T) \,,$$

where T is the total number of iterations to be performed.

- If $\frac{\partial^2 \ell(\mathbf{\Omega}^{(t)};\mathcal{X})}{\partial \mu_{jd}^2} < 0$ or the magnitude of $\delta_t \mu_{jd}$ is too small, backoff to gradient descent by setting the update step as follows:

$$\delta_t \mu_{jd} = -\varepsilon_t \frac{\partial \ell(\mathbf{\Omega}^{(t)};\mathcal{X})}{\partial \mu_{jd}} \,.$$

Step 4: If $|\delta_t \mu_{jd}| > limit \times |\delta_{t-1} \mu_{jd}|$, set $\delta_t \mu_{jd} = \text{sign}(\delta_t \mu_{jd}) \times limit \times |\delta_{t-1} \mu_{jd}|$ to limit the absolute update step size, where $limit$ is a control parameter and set as 1.75 in our experiments.

Step 5: Update μ_{jd} by $\mu_{jd}^{(t+1)} \leftarrow \mu_{jd}^{(t)} + \delta_t \mu_{jd}$.

Step 6: Repeat Step 2 to Step 5 $T - 1$ times.

The formula for calculating $\frac{\partial \ell(\mathbf{\Omega};\mathcal{X})}{\partial \mu_{jd}}$ is omitted here due to space limitation.

4. Compression of PCGM Parameters

According to the discriminant function in Eq. (3), we need to store following three sets of parameters to implement a PCGM-based recognizer:

- the set of transformed mean vectors and constants $\{m_j, c_j\}$, in total $(D+1) \times M$ raw parameters,
- the set of coefficients $\{\lambda_{jl}\}$, in total $L \times M$ raw parameters,
- the set of prototypes $\mathbf{\Psi}$, in total $D(D+1) \times L/2$ raw parameters.

If a 4-byte floating point number is used to represent each raw parameters, the total memory requirement is about $4 \times (D + L + 1) \times M + 4 \times D(D+1) \times L/2$ bytes, which translates into about 2.83MB for a typical system setup of $D = 128$, $L = 32$, $M \approx 3000$.

To compress the above PCGM parametrs, we use the split-VQ technique [7], which is well-known in speech recognition and coding areas, and was also used previously in OCR [8] and handwriting recognition areas (e.g. [5, 6, 13]). For each transformed mean vector $m_j \in R^D$, we first uniformly split it into Q_1 D_{Q_1}-dimensional sub-vectors (i.e., $D = Q_1 \times D_{Q_1}$). Then for each $q \in \{1, \cdots, Q_1\}$, LBG algorithm [11] is used to group the set of sub-vectors $\{m_j^q | j = 1, \cdots, M\}$ into 256 clusters, with Euclidean distance as the distortion measure, where m_j^q is the q-th sub-vector of m_j. After clustering, each m_j^q is represented by the index of its nearest centroid. Each centroid (or codeword), a D_{Q_2}-dimensional vector, is represented by D_{Q_2} 4-byte floating point numbers, while each index is represented as a single-byte unsigned integer. In total, we require $Q_1 \times M$ bytes to store the indices and $4 \times D \times 256$ bytes to store the codebook. Likewise, for the set of coefficients $\{\lambda_{jl}\}$, we first uniformly split each vector $\Lambda_j = (\lambda_{j1}, \cdots, \lambda_{jL})^T$ into Q_2 D_{Q_2}-dimensional sub-vectors (i.e., $L = Q_2 \times D_{Q_2}$). Then, for each $q \in \{1, \cdots, Q_2\}$, group the set of sub-vectors $\{\Lambda_j^q | j = 1, \cdots, M\}$ into 256 clusters, where Λ_j^q is the q-th sub-vector of Λ_j. Therefore, we need $Q_2 \times M + 4 \times L \times 256$ bytes to store the coefficients. For the compression of prototypes $\Psi = \{S_1, \cdots, S_L\}$, since each prototype S_l is symmetric, it is only necessary to store the diagonal and upper-diagonal items. It is noted that the diagonal items reflect the auto-correlations of elements in the feature vector and have a dynamic range significantly different from that of off-diagonal items. Therefore, we quantize all the off-diagonal items using 8-bit scalar quantization, leaving the diagonal items intact. This gives rise to a memory requirement of $D(D-1) \times L/2 + 256 \times 4 + D \times L \times 4$ bytes to store the prototypes. It is also possible to quantize c_j. However, since storing c_j costs only several kilobytes, we just skip the compression of c_j in this study.

A similar model compression strategy is also applied to MQDF-based recognizers. We use an approach slightly different from the one in [13]. Mean vectors are quantized as above by split-VQ algorithm with each sub-vector in D_{R_1}-dimensional space. Each eigenvector v_{jk} is also uniformly split into R_2 D_{R_2}-dimensional sub-vectors. Then for each $q \in \{1, \cdots, R_2\}$, the set of sub-vectors $\{v_{jk}^q | j = 1, \cdots, M; k = 1, \cdots, K\}$ are grouped into 256 clusters, where v_{jk}^q is the q-th sub-vector of the eigenvector v_{jk}. We also compress eigenvalues ρ_{jk} and constants δ_j using 8-bit scalar quantization.

If all the model parameters are updated by MCE training, model compression should be done after the completion of

MCE training. However, because we only update mean vectors in MCE training, we can combine MCE training and model compression as follows:

Step 1: Quantize coefficients $\{\lambda_{jl}\}$ and prototypes Ψ into $\{\hat{\lambda}_{jl}\}$ and $\hat{\Psi}$ by using the above approach.

Step 2: Invoke MCE training to fine-tune the mean vectors $\{\mu_j\}$, where the compressed precision matrix $\hat{P}_j = \sum_{l=1}^{L} \hat{\lambda}_{jl}\hat{S}_l$ is used to calculate the PCGM's discriminant function in Eq. (2).

Step 3: Transform mean vectors, i.e., $m_j = \hat{P}_j \mu_j^*$, and update c_j by $c_j \leftarrow c_j + \overline{\mu}_j^T \hat{P}_j \overline{\mu}_j - (\mu_j^*)^T \hat{P}_j \mu_j^*$, where $\overline{\mu}_j$ and μ_j^* denote the values of the mean vector μ_j before and after MCE training respectively.

Step 4: Quantize the transformed mean vectors $\{m_j\}$ by using the above approach.

A similar strategy is also applied to MCE training and model compression for MQDF-based classifiers.

5. Experiments and Results

5.1. Experimental Setup

In order to evaluate and compare the capability and limitation of different modeling approaches, we conduct a series of experiments on the task of the recognition of isolated online handwritten characters with a vocabulary of 2965 level-1 Kanji characters in JIS standard. The popular Nakayosi and Kuchibue Japanese character databases [15] are used. The Nakayosi database consists of about 1.7 million character samples from 163 writers, and the Kuchibue database contains about 1.4 million character samples from 120 writers. We select randomly about 92% samples from the Nakayosi database to form the training data set, 75% samples from the Kuchibue database to form the testing data set, while the remaining samples from both databases are used to form a development set for tuning control parameters. By this partition, there are 704,650 samples in the training set, 229,398 in the development set, and 506,848 in the testing set, respectively.

As for feature extraction, a 512-dimensional raw feature vector z is first extracted from each handwriting sample by using the procedure described in [2]. Then we use the LDA transformation matrix W estimated from the training data to transform z into a new feature vector x of dimension 128 (e.g., [8]). All of our experiments are conducted on these 128-dimensional feature vectors. To speed up recognition, a multiple-prototype-based pre-classifier is used first to identify a short-list of 50 candidates for each testing sample. After that, relatively more "expensive" MQDF or PCGM-based classifiers are called to choose the top candidate from the short-list.

5.2. Experimental Results

Using the ML-training algorithm developed in [18], we first construct three PCGM-based classifiers, namely PCGM(32), PCGM(64) and PCGM(128), where PCGM(L) means L prototypes are used. Recognition accuracy (in %) on testing set and the corresponding memory requirement (in MB) are summarized in the 2nd and 3rd columns of Table 1 respectively. In calculating the model size, we assume that each model parameter is represented by a 4-byte floating point number.

Next, the precision matrices in PCGMs are compressed using the procedure described in section 4. Although it is possible to use larger D_{Q_2} to generate more compact models, we set $D_{Q_2} = 1$ in our experiments. The corresponding results are summarized in the 4th and 5th columns of Table 1. It is observed that the recognition accuracies only degrade slightly, but the footprints are reduced dramatically.

In the third set of experiments, we carried out MCE training to fine-tune the mean vectors of PCGM-based classifiers. To save computations, we actually used 100 most competing classes, instead of $M - 1$ classes, for each training sample to define $G_j(x; \mathbf{\Omega})$ in Eq. (6). The setting of other control parameters is as follows: $\alpha = 0.1$, $\beta = 0$, $\eta = 0.2$, $\varepsilon_0 = 0.1$, $T = 20$. The corresponding results are summarized in the 6th and 7th columns of Table 1. Compared with ML-trained and precision-matrix-compressed PCGMs, MCE training brings relative error reductions of 17.9%, 13.5%, and 8.0% for PCGM(32), PCGM(64), and PCGM(128), respectively. The power of MCE training is clearly demonstrated.

Finally, the split-VQ algorithm is used to quantize the fine-tuned mean vectors. In our experiments, $D_{Q_1} = 1$ is used. The corresponding results are summarized in the 8th and 9th columns of Table 1. The footprint is further reduced, yet no significant degradation of recognition accuracy is observed due to mean vector compression.

For comparison, we also conducted several sets of experiments for MQDF-based classifiers. We use 20 eigenvectors per character class, which gives a recognition accuracy comparable to the best accuracy achievable by the MQDF approach. First we construct an MQDF-based classifier using ML training. Then we use split-VQ to compress the eigenvectors, and 8-bit scalar quantization to compress eigenvalues $\{\rho_{jk}\}$ and constants $\{\delta_j\}$. To construct MQDF-based classifiers using similar amount of memory as our PCGM-based classifiers, a setting of $D_{R_2} = 4$ has to be used. This leads to a significant degradation in recognition accuracy (error rate increases about 30%). Starting from this ML-trained and compressed MQDF-based classifier, MCE training [12] is conducted to fine-tune the mean vectors. MCE training achieves a relative error reduction of 10.7%. Finally, the mean vectors are quantized using split-VQ with the setting $D_{R_1} = 2$. A minor drop in recognition

Table 1. Summary of experimental results.

Classifiers with different approaches and complexities	Methods for Model Training and Compression							
	ML Training No Compression		+ Precision Matrix Compression		+ MCE Training		+ Mean Vector Compression	
	Accuracy (in %)	Memory (in MB)	Accuracy (in %)	Memory (in MB)	Accuracy (in %)	Memory (in MB)	Accuracy (in %)	Memory (in MB)
PCGM(32)	98.00	2.83	97.99	1.84	98.35	1.84	98.35	0.88
PCGM(64)	98.32	4.20	98.29	2.23	98.52	2.23	98.50	1.27
PCGM(128)	98.53	6.94	98.49	3.00	98.61	3.00	98.59	2.04
MQDF(20)	98.52	30.64	98.04	3.47	98.25	3.47	98.20	2.33

accuracy (from 98.25% to 98.20%) is observed at this step. The corresponding experimental results are summarized in the last row of Table 1.

It is clear from the experimental results that our PCGM-based classifiers can be made very compact yet without incurring significant degradation of recognition accuracy. Our PCGM-based approach can achieve a much better accuracy-memory tradeoff than MQDF-based approach. As pointed out in [18], the main disadvantage of PCGM approach is that PCGM-based classifiers require more computations than that of MQDF. For example, running on a PC with a 3 GHz Pentium-4 CPU and using a short-list of 50 candidates, the average recognition time (in terms of user CPU time) of a compact PCGM-based recognizer (PCGM(32) with 0.88MB memory footprint and 98.35% recognition accuracy) is 2.45 ms per handwriting sample, while that of an MQDF-based recognizer (MQDF(20) with 2.33MB memory footprint and 98.20% recognition accuracy) is about 1.50 ms per sample. Although the PCGM-based recognizer is much slower than the MQDF-based recognizer, it is fast enough for the purpose of designing practical handwriting recognizers on most of today's computational platforms.

6. Conclusion

Given the above results, we conclude that PCGM-based approach offers greater flexibility than MQDF-based approach in striking for a better memory-accuracy tradeoff, therefore could be a good solution to designing practical compact handwriting recognition systems for East Asian languages such as Chinese, Japanese, and Korean.

References

[1] Z.-L. Bai and Q. Huo, "A study on the use of 8-directional features for online handwritten Chinese character recognition," in *Proc. ICDAR-2005*, pp.262-266.

[2] Z.-L. Bai and Q. Huo, "An improved approach to extracting 8-directional features for online handwritten Chinese character recognition," unpublished manuscript, 2005. See also, Z.-L. Bai, *A Study on a Goal Oriented Detection and Verification Based Approach for Image and Ink Document Analysis*, Ph.D. Thesis, The University of Hong Kong, 2006, Chapter 6.

[3] Z.-L. Bai and Q. Huo, "A study of nonlinear shape normalization for online handwritten Chinese character recognition: dot density vs. line density equalization," in *Proc. ICPR-2006*, pp.921-924.

[4] S. E. Fahlman, "An empirical study of learning speed in back-propragation networks," Technical Report, CMU-CS-88-162, Carnegie Mellon University, 1988.

[5] Y. Ge and Q. Huo, "A comparative study of several modeling approaches for large vocabulary offline recognition of handwritten Chinese characters," in *Proc. ICPR-2002*, pp.III-85-88.

[6] Y. Ge and Q. Huo, "A study on the use of CDHMM for large vocabulary offline recognition of handwritten Chinese characters," in *Proc. 8th IWFHR*, 2002, pp.334-338.

[7] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*, Springer, 1991.

[8] Q. Huo, Y. Ge, and Z.-D. Feng, "High performance Chinese OCR based on Gabor features, discriminative feature extraction and model training," in *Proc. ICASSP-2001*, pp.III-1517-1520.

[9] B.-H. Juang, W. Chou and C.-H. Lee, "Minimum classification error rate methods for speech recognition," *IEEE Trans. on Speech and Audio Processing,* vol.5, no.3, pp.257-265, 1997.

[10] F. Kimura, K. Takashina, S. Tsuruoka, and Y. Miyake, "Modified quadratic discriminant functions and the application to Chinese character recognition," *IEEE Trans. on PAMI*, vol. 9, pp.149-153, 1987.

[11] Y. Linde, A. Buzo, R. Gray, "An algorithm for vector quantizer design", in *IEEE Trans. on Communications,* vol. 28, pp.84-94, 1980.

[12] C.-L. Liu, H. Sako, and H. Fujisawa, "Discriminative learning quadratic discriminant function for handwriting recognition," *IEEE Trans. on Neural Networks*, vol. 15, no. 2, pp.430-444, 2004.

[13] T. Long and L.-W. Jin,"Building compact MQDF classifier for large character set recognition by subspace distribution sharing," *Pattern Recognition*, vol. 41, no. 9, pp.2916-2925, 2008.

[14] E. McDermott, T. J. Hazen, J. Le Roux, A. Nakamura and S. Katagiri, "Discriminative training for large vocabulary speech recognition using minimum classification error," *IEEE Trans. on Audio, Speech and Language Processing*, vol 15, no 1, pp.203-223, 2007.

[15] M. Nakagawa and K. Matsumoto, "Collection of on-line handwritten Japanese character pattern databases and their analysis," *International Journal on Document Analysis and Recognition*, vol. 7, pp.69-81, 2004.

[16] Y.-Q. Wang and Q. Huo, "A study of semi-tied covariance modeling for online handwritten Chinese character recognition," in *Proc. ICPR-2008.*

[17] Y.-Q. Wang and Q. Huo, "Modeling inverse covariance matrices by expansion of tied basis matrices for online handwritten Chinese character recognition," in *Proc. International Conference on Frontiers in Handwriting Recognition*, 2008, pp.284-289.

[18] Y.-Q. Wang and Q. Huo, "Modeling inverse covariance matrices by expansion of tied basis matrices for online handwritten Chinese character recognition," Pattern Recognition, 2009, in press.