IJDAR (2008) 11:127–141
DOI 10.1007/s10032-008-0073-5

ORIGINAL PAPER

Arabic handwritten digit recognition

Sherif Abdleazeem · Ezzat El-Sherif

Received: 3 March 2008 / Revised: 27 September 2008 / Accepted: 5 October 2008 / Published online: 20 November 2008

Abstract In this paper, we fill a gap in the literature by studying the problem of Arabic handwritten digit recognition. The performances of different classification and feature extraction techniques on recognizing Arabic digits are going to be reported to serve as a benchmark for future work on the problem. The performance of well known classifiers and feature extraction techniques will be reported in addition to a novel feature extraction technique we present in this paper that gives a high accuracy and competes with the state-of-the-art techniques. A total of 54 different classifier/features combinations will be evaluated on Arabic digits in terms of accuracy and classification time. The results are analyzed and the problem of the digit '0' is identified with a proposed method to solve it. Moreover, we propose a strategy to select and design an optimal two-stage system out of our study and, hence, we suggest a fast two-stage classification system for Arabic digits which achieves as high accuracy as the highest classifier/features combination but with much less recognition time.

Keywords Benchmark · Arabic digits · Indian digits · Classifiers · Feature extraction · Two-stage

1 Introduction

Handwritten digit recognition problem can be seen as a subtask of the more general Optical Character Recognition (OCR) problem. However, there are some applications (e.g., postal code and bank checks reading) that are restricted to recognizing digits but require very high accuracy and speed. In addition, handwritten digit recognition problem is usually used as a benchmark for comparing different classification techniques [1].

While recognition of handwritten Latin digits has been extensively investigated using various techniques [1–8], little work has been done on Arabic handwritten digit recognition. Al-Omari et al. [9] proposed a system for recognizing Arabic digits from '1' to '9'. They used a scale-, translation-, rotation-invariant feature vector to train a probabilistic neural network (PNN). Their database was composed of 720 digits for training and 480 digits for testing written by 120 persons. They achieved 99.75% accuracy. Said et al. [10] used pixel values of the 16×20 size-normalized digit images as features. They fed these values to an Artificial Neural Network (ANN), where number of its hidden units is determined dynamically. They used a training set of 2400 digits and a testing set of 200 digits written by 20 persons to achieve 94% accuracy. In a previous paper [11], we introduced a large Arabic Digits dataBase (the ADBase—see Sect. 2.1 for more details) and devised a two-stage system for recognizing Arabic digits. The first stage is an ANN fed with a short but powerful feature vector to handle easy-to-classify digits. Ambiguous digits are rejected to the more powerful second stage which is an SVM fed with a long feature vector. The system had a good timing performance and achieved 99.15% accuracy on the ADBase. Note that results of different works cannot be compared because the used databases are not the same.

Naming conventions for different numeral systems may be confusing. Digits used in Europe and several other countries sometimes are called "Arabic Numbers"; and digits used in Arab world are sometimes called "Hindi Numbers". A different naming convention is used in this paper. Digits used in Europe will be referred to as "Latin Digits" and that

S. Abdleazeem · E. El-Sherif (✉)
Electronics Engineering Department,
American University in Cairo (AUC), Cairo, Egypt
e-mail: ezzatali@aucegypt.edu

S. Abdleazeem
e-mail: shazeem@aucegypt.edu

Table 1 Arabic printed and handwritten digits

Latin Equivalent	0	1	2	3	4	5	6	7	8	9
Printed	٠	١	٢	٣	٤	٥	٦	٧	٨	٩
Typical Handwritten	٠	١	٢	٣	٤	٥	٦	٧	٨	٩
Other Writing Style	--	--	--	٣	--	--	--	--	--	--

Table 2 Persian printed and handwritten digits

Latin Equivalent	0	1	2	3	4	5	6	7	8	9
Printed	۰	۱	۲	۳	۴	۵	۶	۷	۸	۹
Typical Handwritten	۰	۱	۲	۳	۴	۵	۶	۷	۸	۹
Other Writing Style	٠	--	٢	٣	٤	٥	٦	--	--	--

used in Arab world as "Arabic Digits". It is worthwhile to mention here that Arabic and Persian handwritten digits (digits used in Iran) are similar but not identical. However, there are some writing styles for Persian digits that are very similar to Arabic which leads some researchers to consider Arabic and Persian digits to be the same [12,13]. Tables 1 and 2 show Arabic and Persian handwritten digits with different writing styles as well as their printed versions.

In this paper, we are going to study the performance of various classifiers/features combinations on the Arabic digit recognition problem. Well-known feature extraction techniques besides one novel technique we introduce in this paper are considered. Results are then analyzed leading to the notice of the problem introduced by the Arabic digit '0'. A suggestion of how to alleviate this problem is then introduced. For the sake of comparison, the performances of the same classifier/features combinations are evaluated on Latin digits. Moreover, a selection process of a two-stage system is presented and an optimal two-stage system for Arabic digits is suggested.

The remaining of the paper is organized as follows. Section 2 is about the two Arabic digits databases used in this study: the ADBase and the MADBase. Sects. 3 and 4 introduce the classification and the feature extraction techniques used, respectively. Section 5 reports and discusses the results. Section 6 is about the selection process of the optimal two-stage classifier for Arabic digits. And in Sect. 7, we conclude.

2 Arabic digits databases

Both databases (ADBase and MADBase) are available for free online at http://datacenter.aucegypt.edu/shazeem.

2.1 The ADBase

The ADBase [11] is composed of 70,000 digits written by 700 participants. Each participant wrote each digit (from '0' to '9') ten times. The database is partitioned into two sets: a training set (60,000 digits to 6,000 images per class) and a test set (10,000 digits to 1,000 images per class). Writers of training set and test set are exclusive. Ordering of including writers to test set are randomized to make sure that writers of test set are not from single institution (to ensure variability of the test set).

2.2 The MADBase

The MADBase is a modified version of the ADBase that has the same format as MNIST [1]. This is done to ensure the validity of any comparison made between Latin and Arabic digit recognition problems (see Sect. 5 for a comparison between Arabic and Latin digits results). The MADBase is created from ADBase as follows. For each digit of ADBase, its height (h) and width (w) are calculated, and then size-normalized [20] to have a new height (h_{new}) and new width (w_{new}). The assigned values of h_{new} and w_{new} depend on whether h or w is greater. If $h > w$, then h_{new} is set to 20, and w_{new} to floor $(20 \times w/h)$. If $w > h$, then w_{new} is set to 20 and h_{new} to floor $(20 \times h/w)$. This procedure ensures that each digit of MADBase is confined in a 20×20 box, while its aspect ratio is preserved. Then each digit is placed in a 28×28 white background such that its center of gravity coincides with the center of the white background.

Note that the images of the ADBase are binary. When the ADBase images are down-sampled to form the MADBase, an antialiasing filter is applied to the images. This made the images of the MADBase gray-scaled.

Figure 1 shows samples of ADBase and their MADBase versions. In this paper, we evaluate the performance of different classification and feature extraction techniques on MADBase. The ADBase is used just for extracting size information required to alleviate the problem of the Arabic digit '0' as will be clear in Sect. 5.

3 Brief description of the used classification techniques

In this section, a brief description of each of the used classification techniques is going to be presented. In the results section (Sect. 5), the accuracy of each classifier/features combination and the timing performance of each classifier will be reported as well. Some of the used classification techniques have parameters that need to be specified (e.g. number of hidden neurons in the neural network). Such parameters are optimized using a validation set. The validation set is composed of 10,000 samples chosen randomly from the training

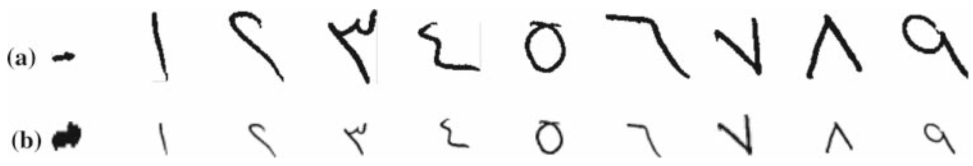


Fig. 1 Sample digits from ‘0’ to ‘9’ from **a** ADBase, **b** the corresponding digits in MADBase

set. Once the optimized parameters values are found, they are used to train the classifiers on the whole training set.

3.1 K-Nearest neighbor

The K-Nearest Neighbor (KNN) [14] is one of the simplest classification techniques; yet gives surprisingly high accuracy. In KNN, there is no training stage. All training samples must be present in the testing phase; and Euclidean distances between each training sample and the sample to be tested are calculated. The K training samples that have the smallest distances to the test sample are found and their classes are identified. The most frequent class in the selected K training samples is declared to be the class of the test sample. In this study, we used $K = 3$.

3.2 Parzen window

Parzen window estimates the probability density for the input feature space using kernel density estimation technique [15]. In this technique, every sample in the training set creates a bump in the feature space with a shape that depends on the type of kernel used. For a certain test sample, the probability that it belongs to a certain class is the sum of all the contributions of bumps created by the training samples of that class at this test sample. The kernel used in this study is the Gaussian kernel with a variance chosen to have the best performance on the validation set. The Parzen window, like KNN, needs all the training set to be available at test time.

3.3 Two-layer neural network

The Two-Layer Neural Network (2-NN) [14] is one of the most powerful classifiers. It can actually model any target function if we are allowed to increase the number of hidden units as we wish. Unfortunately, an ANN with very large hidden layer trained by limited number of training samples gives poor results because of the overfitting problem. Hence, the number of hidden units of ANN should be selected to be high enough to model the problem at hand but not too high to overfit. The number of hidden units is selected to have the best performance on the validation set.

3.4 OVO One-Layer Neural Network

One-Layer Neural Network (1-NN) is a linear classifier that originally solves two-class problems. One way to extend linear classifiers to the C-class case (where $C = 10$ in case of our problem) is to train C different two-class linear classifiers; each responsible for separating one class from the other $(C - 1)$ classes. This technique is called “One-versus-All” (OVA) learning. Another approach is the so-called “One-versus-One” (OVO) or “pairwise” approach [16]. In this technique, $C(C - 1)/2$ classifiers (45 classifiers in our case) are trained. Each classifier is trained using the training samples of just two classes; i.e., each classifier specializes in separating just two classes. In the test phase, each test sample is presented to all the $C(C - 1)/2$ classifiers; and the most frequently decided class is declared the winner. An OVO linear classifier can construct decision boundaries that cannot be constructed using an OVA linear classifier [17,18]; hence, it is more powerful.

3.5 PCA + Quadratic

Linear classifiers assume that decision boundaries needed for classification can be constructed using a set of hyperplanes. Such an assumption may or may not hold. If this assumption does not hold and we still wish to use a linear classifier, we may transform the problem into a space of much higher dimension; high enough for the problem to be linearly separable. One way to increase the dimensionality of the problem is to use a polynomial expansion of the input vector. Unfortunately, this expansion leads to prohibitively large dimensionality especially if the problem original dimensionality is already high. One way to overcome this problem is to select only the most informative dimensions of the original problem and ignore the others. Most informative dimensions (in the original space or a linearly transformed version of it) can be found using Principal Component Analysis (PCA) algorithm [15]. In this work, we use PCA to extract the most informative dimensions and then quadratically expand them [1,3]. The quadratically expanded feature vector is then fed to 1-NN. We denote this technique PCA + Quadratic.

3.6 OVO linear SVM

SVM is considered one of the most powerful classification techniques and is now widely used in many pattern recognition applications. The linear SVM is originally a binary classifier that constructs a hyperplane separating between two classes just like 1-NN. However, SVM algorithm tries to find the most optimal separating hyperplane; the one that achieves maximum margin [19]. To extend the linear SVM to handle the multiclass case ($C > 2$) we used the OVO technique discussed earlier. We add here that the linear SVM could be seen as an SVM with a dot product kernel $K(\mathbf{x}_1, \mathbf{x}_2)$ [19],

$$K(\mathbf{x}_1, \mathbf{x}_2) = s(\mathbf{x}_1 \bullet \mathbf{x}_2), \tag{1}$$

where s is constant, we chose to be 1.

3.7 OVO RBF SVM

The linear SVM assumes that classes to be separated are linearly separable. If this assumption does not hold, then we can use the idea of space dimensionality expansion discussed earlier. Because the expanded space would be of very high dimensionality, the feature dimensionality might be reduced first using PCA. However, non-linear SVM uses a mathematical trick that enable us to expand (using a kernel) the feature vector to any dimensionality we desire (even infinite dimensionality!) without actually the expansion process [19]. The nonlinear SVM is originally designed for two-class problem. Extending it to multi-class can be done using the OVO or the OVA schemes. We used the OVO scheme. [start delete] We evaluated only the unique support vectors to optimize the recognition time [end delete]. The kernel we used is the RBF kernel $K(\mathbf{x}_1, \mathbf{x}_2)$,

$$K(\mathbf{x}_1, \mathbf{x}_2) = \exp\left(-\gamma \left\|\mathbf{x}_1 - \mathbf{x}_2\right\|^2\right), \tag{2}$$

where the γ is the RBF kernel parameter.

To classify a pattern $\mathbf{x}$ using OVO RBF SVM, the following discriminant function $g_{nm}(\mathbf{x})$ is evaluate for each class pair of classes n and m,

$$g_{nm}(\mathbf{x}) = \sum_{i=1}^{N_{nm}} \alpha_{nmi}\, y_{nmi}\, K(\mathbf{x}, \mathbf{x}_{nmi}) + b, \tag{3}$$

where $\mathbf{x}_{nmi}$ is the ith pattern in the subset of the training set containing classes n and m, α_{nmi}'s are constants found by the training algorithm of SVM, and y_{nmi}'s are constants such that $y_i = +1$ when $\mathbf{x}_{nmi}$ belongs to class n and $y_i = -1$ when $\mathbf{x}_{nmi}$ belongs to class m. The constants α_i's have nonzero values for only a subset of the training set called "support vectors". Eq. (3) then needs not be calculated for all training set patterns; only support vectors need to be considered. The number of support vectors then is an important factor affecting the evaluation time of the nonlinear SVM. Actually, there are many situations in which we find that a certain training pattern is used as a support vector in more than one binary classifier of the OVO scheme. Hence, to save recognition time, we apply the kernel for a certain support vector for some pair of classes only once. If the same support vector is used with another pair of classes, its previously calculated kernel evaluation is used. The common support vectors are found as follows. We prepare a list that is initially empty to hold the common support vectors (we will denote it **CSV**). After training a binary SVM that separates between digits n and m, it will have a list of support vectors $\mathbf{SV}_{nm}$. Each element of $\mathbf{SV}_{nm}$ is considered; if it is already in **CSV**, we ignore it and continue scanning $\mathbf{SV}_{nm}$; if not, we add it to **CSV**. We do this for all binary SVMs. While doing such scanning of support vectors, we also replace the list of support vectors of each binary SVM, $\mathbf{SV}_{nm}$, with a new list that contains a pointer for each support vector to its location in **CSV**; we call this list $\mathbf{PSV}_{nm}$. Now, when a pattern $\mathbf{x}$ is required to be classified by the OVO SVM scheme, we calculate the kernel evaluations for this pattern using each common support vector in **CSV**, and then put them in a list we call **KCSV** with the same order of common support vector in **CSV**. Now we are done with all the necessary kernel evaluations. To calculate the value of Eq. 3 for the pattern $\mathbf{x}$, we scan the $\mathbf{PSV}_{nm}$ in each binary SVM and grab the corresponding previously calculated kernel evaluations from **KCSV**.

3.8 Gaussian classifier

Gaussian classifier (GC) starts with the assumption that the probability distribution of feature vector of each class is Gaussian. This leads to the following discriminant function for class c [14],

$$g_c(\mathbf{x}) = \mathbf{x}^t \mathbf{W}_c \mathbf{x} + \mathbf{w}_c^t \mathbf{x} + w_{c0}, \tag{4}$$

where $\mathbf{W}_c = -\frac{1}{2}\boldsymbol{\Sigma}_c^{-1}$, $\mathbf{w}_c = \Sigma_c^{-1}\boldsymbol{\mu}_c$, $w_{c0} = -\frac{1}{2}\boldsymbol{\mu}_c^t \boldsymbol{\Sigma}_c^{-1}\boldsymbol{\mu}_c - \frac{1}{2}\ln|\boldsymbol{\Sigma}_c|$, $\boldsymbol{\mu}_c$ is the estimated mean of $\mathbf{x}$ that belongs to class c, $\boldsymbol{\Sigma}_c$ is the estimated covariance matrix of $\mathbf{x}$ that belongs to class c, and 't' denotes matrix transpose. The parameters $\boldsymbol{\mu}_c$ and $\boldsymbol{\Sigma}_c$ are estimated for each class in the training phase using any parameter estimation technique like maximum likelihood which we used. We note that the size of the matrix $\boldsymbol{\Sigma}_c$ is $d \times d$; this means that the number of parameters to be estimated is a function of d^2. For large feature vector, this leads to overfitting and the inverse of $\boldsymbol{\Sigma}_c$ becomes singular. To avoid this, we may put some restrictions on the form of the covariance matrix, e.g. all classes have the same $\boldsymbol{\Sigma}$. Another option is to reduce the dimensionality of the problem, e.g. using PCA. We have taken the latter approach. The dimension of the feature vector after applying PCA (d') is specified using the validation set. Noting that the quantities $\mathbf{W}_c$, $\mathbf{w}_c^t$, and w_{c0} can be calculated off-line.

3.9 Fisher linear discriminant

Fisher linear discriminant technique [15] separates two classes using a linear hyperplane defined by the equation, $\mathbf{w}^t\mathbf{x} + w_0 = 0$, where $\mathbf{w} = \mathbf{S}_W^{-1}(\boldsymbol{\mu}_1 - \boldsymbol{\mu}_2)$, $\mathbf{S}_W = \frac{1}{2}(\boldsymbol{\Sigma}_1 + \boldsymbol{\Sigma}_2)$, $\boldsymbol{\mu}_1$ is the first class mean, $\boldsymbol{\mu}_2$ is the second class mean,

$\mathbf{\Sigma}_1$ is the first class covariance matrix, $\mathbf{\Sigma}_2$ is the second class covariance matrix, and $w_0 = -\frac{1}{2}(\mu_1 + \mu_2)\mathbf{S}_W^{-1}(\mu_1 - \mu_2)$. The technique is extended to handle multiclass using OVO scheme described in Sect. 3.4.

4 Brief description of the used feature extraction techniques

In this comparative study, we use a set of well-known feature extraction techniques. In addition, we introduce a novel feature extraction technique we called "local directional features" that gives high accuracy rate compared with other features. In this section, a brief description of the used feature extraction techniques is introduced.

All feature extraction techniques that will be discussed operate on the 20 × 20 window of the MADBase image that confine the digit; i.e. after removing the blank border around it. After applying the feature extraction algorithm on a digit image, a variable transformation ($x^{0.5}$) is then applied on the resulting feature vector [3].

4.1 Gradient features

To extract gradient features [3], the gradient operator is first applied to the gray-scale image of the digit to give two gradient components: strength $|\mathbf{g}(x, y)|$ and direction $\angle\mathbf{g}(x, y)$ at each point (x, y) of the image $\mathbf{f}$. This is done by applying Sobel operator [20] on the image to extract vertical and horizontal gradient components,

$$g_x(x, y) = \mathbf{f}(x+1, y-1) + 2\mathbf{f}(x+1, y) + \mathbf{f}(x+1, y+1) \\ -\mathbf{f}(x-1, y-1) - 2\mathbf{f}(x-1, y) - \mathbf{f}(x-1, y+1), \quad (5)$$

$$g_y(x, y) = \mathbf{f}(x-1, y+1) + 2\mathbf{f}(x, y+1) + \mathbf{f}(x+1, y+1) \\ -\mathbf{f}(x-1, y-1) - 2\mathbf{f}(x, y-1) - \mathbf{f}(x+1, y-1), \quad (6)$$

where $\mathbf{f}(x, y)$ is the intensity of image $\mathbf{f}$ at point (x, y), $g_x(x, y)$ is the gradient component at x-direction at location (x, y), and $g_y(x, y)$ is the gradient component at y-direction at location (x, y).

Then the gradient strength and direction is extracted using Eqs. (7) and (8), respectively,

$$|\mathbf{g}(x, y)| = \sqrt{g_x^2(x, y) + g_y^2(x, y)} \quad (7)$$

$$\angle\mathbf{g}(x, y) = \tan^{-1}\left(\frac{g_y(x, y)}{g_x(x, y)}\right) \quad (8)$$

The gradient vector $g(x, y)$ (expressed as strength [delete]strength[delete] $|\mathbf{g}(x, y)|$ and direction $\angle\mathbf{g}(x, y)$) at each point (x, y) of the image is then decomposed into the eight Freeman [20] directions shown in Fig. 2. The gradient vector is decomposed into the eight Freeman directions by projecting the vector into the nearest two Freeman directions as shown in Fig. 3.

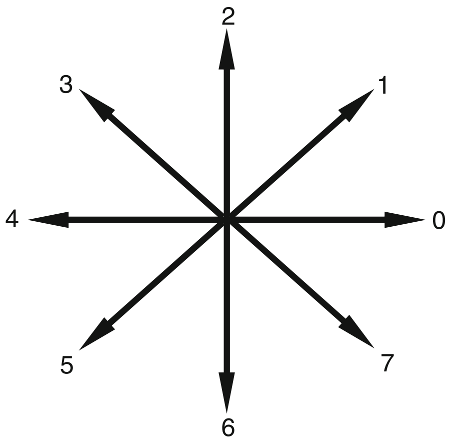


Fig. 2 Freeman directions

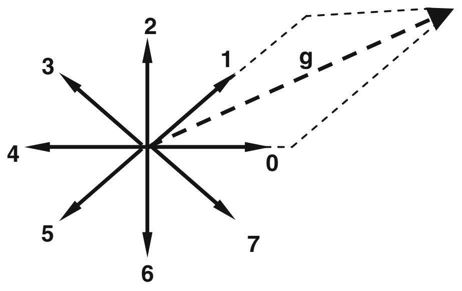


Fig. 3 Projecting the gradient vector **g** into the two nearest Freeman directions

The gradient features are composed of eight layers; each corresponding to one of the Freeman directions. Each layer is the projection of the gradient vectors of the image into the corresponding Freeman direction.

A Gaussian mask $h(x, y)$ is then applied to each layer,

$$h(x, y) = \frac{1}{2\pi\sigma^2}\exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right), \quad (9)$$

and then the layers are uniformly sampled to give 5 × 5 measurements. The parameter σ is related to the sampling interval t via the empirical relation$\sigma = \frac{\sqrt{2}t}{\pi}$ [3]. This means that the gradient features are composed of $8 \times 5 \times 5 = 200$ elements. The gradient feature extraction technique is very powerful as shown in [3] and as will be shown in Sect. 5 of this paper.

4.2 Kirsch features

The Kirsch features [3] are extracted by decomposing the image into 4 layers corresponding to four edge orientations: horizontal ($\mathbf{g}_h$), vertical ($\mathbf{g}_v$), right diagonal ($\mathbf{g}_{rd}$), and left diagonal ($\mathbf{g}_{ld}$). These layers are extracted by applying Kirsch

$\mathbf{g}_{h1}$

5	5	5
-3	0	-3
-3	-3	-3

$\mathbf{g}_{h2}$

-3	-3	-3
-3	0	-3
5	5	5

$\mathbf{g}_{v1}$

5	-3	-3
5	0	-3
5	-3	-3

$\mathbf{g}_{v2}$

-3	-3	5
-3	0	5
-3	-3	5

$\mathbf{g}_{rd1}$

-3	-3	-3
5	0	-3
5	5	-3

$\mathbf{g}_{rd2}$

-3	5	5
-3	0	5
-3	-3	-3

$\mathbf{g}_{ld1}$

-3	-3	-3
-3	0	5
-3	5	5

$\mathbf{g}_{ld2}$

5	5	-3
5	0	-3
-3	-3	-3

Fig. 4 Kirsch masks

masks on the image,

$$\mathbf{g}_h = \max\left(|\mathbf{f} * \mathbf{k}_{h1}|, |\mathbf{f} * \mathbf{k}_{h2}|\right), \quad (10)$$

$$\mathbf{g}_v = \max\left(|\mathbf{f} * \mathbf{k}_{v1}|, |\mathbf{f} * \mathbf{k}_{v2}|\right), \quad (11)$$

$$\mathbf{g}_{rd} = \max\left(|\mathbf{f} * \mathbf{k}_{rd1}|, |\mathbf{f} * \mathbf{k}_{rd1}|\right), \quad (12)$$

$$\mathbf{g}_{ld} = \max\left(|\mathbf{f} * \mathbf{k}_{ld1}|, |\mathbf{f} * \mathbf{k}_{ld2}|\right), \quad (13)$$

where * denotes 2D convolution operation [20], $\mathbf{k}_{h1}$ and $\mathbf{k}_{h2}$ are Kirsch masks responsible for extracting horizontal edges, $\mathbf{k}_{v1}$ and $\mathbf{k}_{v2}$ for vertical edges, $\mathbf{k}_{rd1}$ and $\mathbf{k}_{rd2}$ for right diagonal edges, and $\mathbf{k}_{ld1}$ and $\mathbf{k}_{ld2}$ for left diagonal edges. Figure 4 shows all Kirsch masks.

A Gaussian mask is then applied to all the 4 layers and 5×5 measurements are extracted from each of them in the same manner done with gradient features (Sect. 4.1). This means that Kirsch features are composed of $4 \times 5 \times 5 = 100$ elements.

4.3 Local chain code features

To extract the local chain code features [21], the contour of the digit is first followed by a contour following algorithm [31], and then each contour point of the image is marked with the corresponding Freeman direction. The image is then decomposed into 8 layers; each layer contains only contour points that belong to the corresponding Freeman direction. Each of the eight layers is then uniformly partitioned into 5×5 zones. Each zone is averaged leading to a feature vector of 200 elements.

4.4 Wavelet features

Wavelet transform [20] extracts the essential information contained in a signal or image. This may help in reducing the dimensionality of the problem which leads to faster classification. Also it might reduce the noise and the non-essential information that may confuse the classifier leading to better classification performance. Wavelet transform could be applied on the image directly or could be applied on the gradient decomposition of the image [22]. In our study, we applied the former method. The image is first resized to be 64×64 and then the image is composed into three resolution levels. The third level approximation of the image (which is a 8×8 image) is then used as features. This means that wavelet features are composed of 64 elements.

4.5 Concatenation of low-dimensional features

Here we concatenate six families of low-dimensional features to form one feature vector. While each of these low-dimensional features is not very effective, their concatenation proved to be a powerful feature vector. Each of the six low-dimensional feature families is going to be discussed in the following subsections.

4.5.1 Raw image zoning

In this technique, the image is uniformly partitioned into 5×5 zones [23]. The average of each zone is calculated leading to a feature vector of 25 elements. See Fig. 5.

4.5.2 Vertical and horizontal projections

In this technique, the horizontal and vertical histograms of the image are calculated leading to a feature vector of 40 elements [23] (remember that we are working on the 20×20 window that confines the digit not all 28×28 pixels of the MADBase digits). See Fig. 6.

4.5.3 Vertical and horizontal cross counts

The image is first binarized, and then the vertical and horizontal cross counts are calculated. The vertical cross counts are calculated by scanning each row of the binary image and each transition from 0 to 1 or from 1 to 0 increases a counter that has an initial value of zero; then the scanned row is associated with the counter value at the end of the scanning process. A similar procedure is done for each column. This leads to a feature vector of 40 elements. See Fig. 7.

4.5.4 Centroidal distances

In this technique [11,24], the digit image is partitioned into 16 sectors around the digit center of gravity (the centroid) as shown in Fig. 8. Then the centroid of each sector is calculated. The distances between the whole digit centroid and sectors centroids are calculated and normalized by dividing them by the digit bounding box diameter. This forms a 16-element feature vector.

4.5.5 Directional features

The directional features are formed as follows [11,25]. Each background pixel is labeled by a 4-element vector. For each

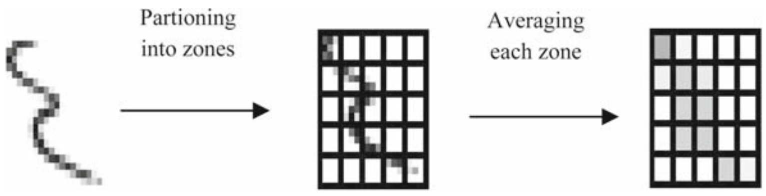


Fig. 5 Illustrating how raw image zoning features are formed (refer to Sect. 4.5.1)

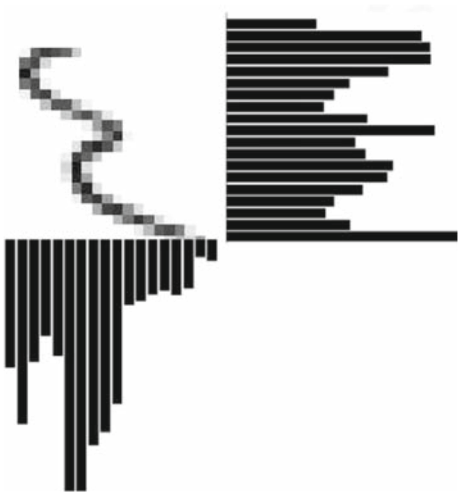


Fig. 6 Illustrating how projections features are formed (refer to Sect. 4.5.2)

background pixel we walk upward; if a foreground pixel is found, the first element of the 4-element vector is set to '1' otherwise set to '0'. Then we walk to the right; if a foreground pixel is found, the second elements is set to '1' otherwise set to '0'; and so on till we finish the four principal directions. Then for each combination of 0's and 1's in the 4-element vector, we give a different label. Hence, we have 16 different labels.

The label '0000' that indicates finding no foreground pixel in any direction actually does not show up because the image border bound the digit either in the horizontal or the vertical direction. Hence, we have only 15 labels. Now each of background pixels has one label of the 15 labels. The final feature vector is the histogram of such labels giving rise to a 15-element feature vector.

4.5.6 Length-normalized contour

In this technique [11], the horizontal and vertical coordinates of the boundary pixels of the digit are used as features. First, the boundary of the digit is extracted using a contour following algorithm; then, each boundary point is stored by its horizontal and vertical coordinates. Horizontal locations are magnitude-normalized by dividing them by the width of the digit bounding box. Vertical locations are also magnitude-normalized by dividing them by the height of the bounding

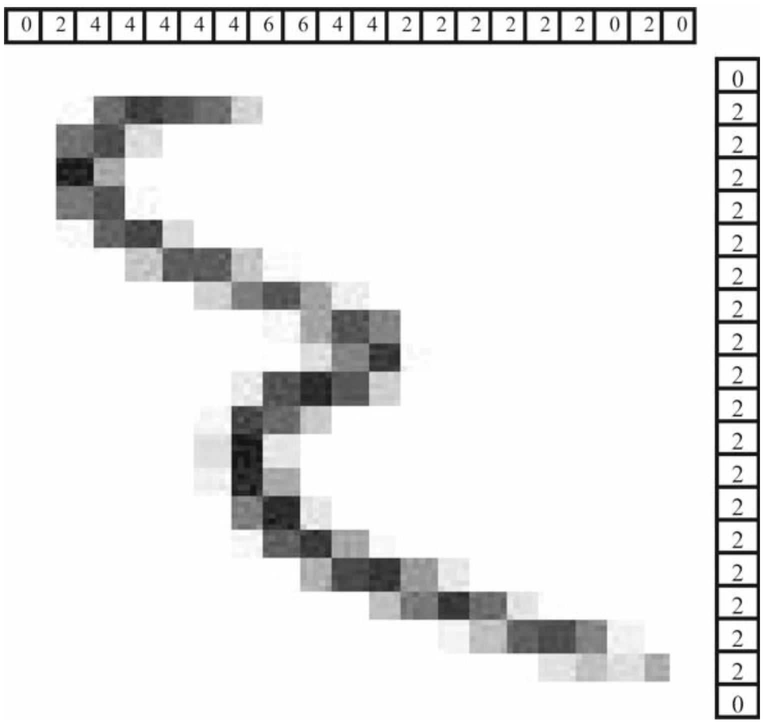


Fig. 7 Illustrating how cross counts features are formed (refer to Sect. 4.5.3)

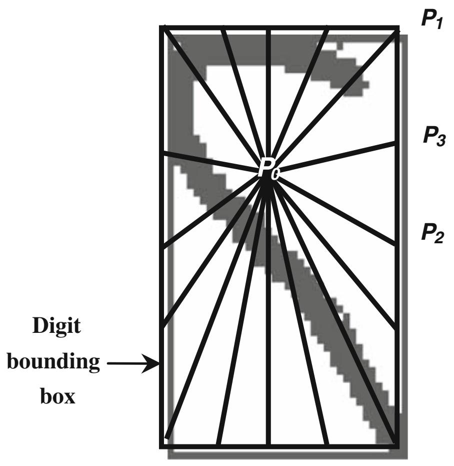


Fig. 8 Forming the 16 sectors required to calculate the centroidal distances features (P_0 is the *centroid* of the whole digit, P_1 is the *upper right corner* point, P_2 is the *mid-point* of the left edge of the bounding box, and P_3 is the *mid-point* between P_1 and P_2)

box. Then both horizontal and vertical locations are partitioned into 10 portions and the average of each portion is calculated leading to a feature vector of length 20.

Feeding the classifier with the concatenation of different features without any preprocessing step would lead to very bad results because features of large magnitudes will dominate. Hence, we applied mean and variance normalization [14] to all the low-dimensional features discussed in this section before concatenating them into a feature vector of length 156.

4.6 Local directional features

Local directional features extraction is a novel technique that, we present in this paper. It is a natural extension of the directional features presented in Sect. 4.5.5. Like directional features, each background pixel is labeled by a 4-element vector which leads to 16 different labels. However not all labels are used. Only nine labels corresponding to 9 situations are considered: (1) closed from all directions (i.e. black pixel can be reached when moving in all the principal four directions), (2) open up (i.e. black pixel can be reached when moving in the principal four directions except when moving upward), (3) open down, (4) open right, (5) open left, (6) open up and right, (7) open up and left, (8) open down and right, and (9) open down and left.

The image is then decomposed into nine layers; each layer corresponds to one of the nine labels. Pixel (x, y) in layer $\#n$ is illuminated only if pixel (x, y) in the image has been given the label $\#n$. A Gaussian mask is then applied to all the nine layers and 5×5 measurements are extracted from each of them in the same manner done with gradient features (Sect. 4.1). This means that local directional features are composed of 225 elements. See Fig. 9.

5 Results

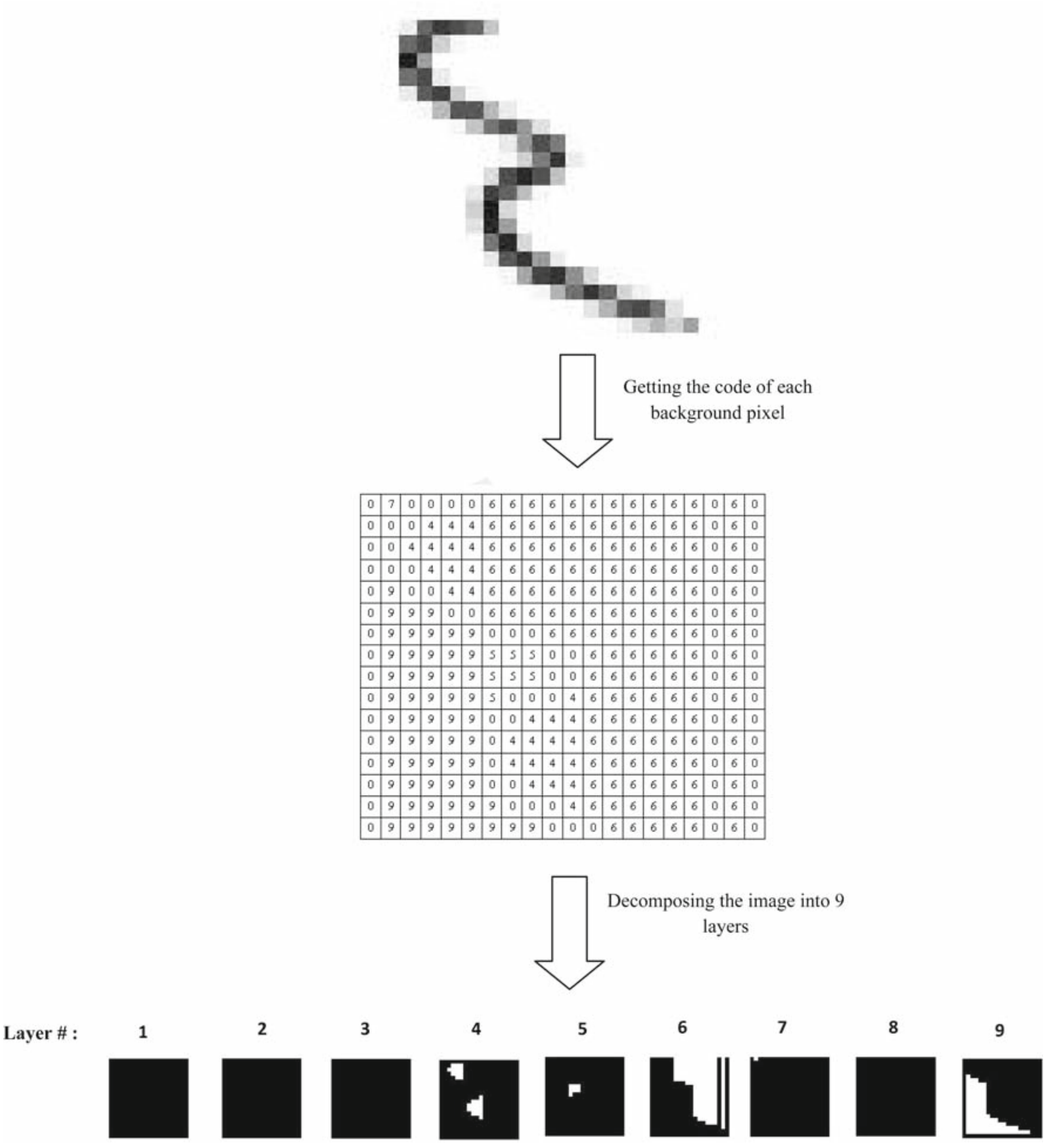
In this section, the accuracies of each pair of the nine classifiers and six feature extraction techniques on the Arabic digit recognition are going to be presented. Many of the classification techniques used have some parameters that need to be adjusted. Here we summarize these parameters and discuss how we set their values:

(i) *Number of hidden neurons of the neural networks (h).*
On a set composed of {training set}-{validation set} (the validation set is composed of 10,000 patterns selected randomly from the training set), ANNs are trained with the following numbers of hidden neurons $h = \{50, 100, 150, 200, 250, 300\}$. The value of h that gives the highest accuracy on the validation set is chosen.

(ii) *The constant C of the linear SVM.*
On the set {training set}-{validation set}, we tried $C = \{0.1, 1, 10, 100\}$. The best performing value on the validation set is then chosen.

(iii) *The constants C and γ of the RBF SVM.*
The usual procedure taken to select RBF SVM parameters is by trying all the possible combinations of C and γ [26]. However, this procedure is very time consuming especially if we take into consideration the large number of experiments in this comparative study. Hence, we took an alternate procedure. First we set $C = 10$ which is, based on our experience, is a good choice for all the features we used. With this value of C, we try $\gamma = \{0.01, 0.1, 1, 10\}$ on the validation set. For the value of γ that gives maximum accuracy (which we denote $\gamma_{\max\_1}$), we try $C = \{0.1, 1, 10, 100\}$. We then select the best C which we denote $C_{\max\_1}$. This is considered a first stage in the process of searching for the best C and γ. In the second stage we try $\gamma = \{\gamma_{\max\_1}/3, \gamma_{\max\_1}/2, 2\gamma_{\max\_1}, 3\gamma_{\max\_1}\}$. The best of those we denote $\gamma_{\max\_2}$ and with which we try $C = \{C_{\max1}/3, C_{\max1}/2, 2C_{\max1}, 3C_{\max1}\}$. The best C is then denoted $C_{\max\_2}$. This is the second stage of the search process. We may add as many stages as we wish to fine tune the value of C and γ. In our work, we stopped at the second stage, i.e. chose $C = C_{\max\_2}$, and $\gamma = \gamma_{\max\_2}$.

(iv) *The kernel variance of Parzen window (σ^2).*
On the set {training set}-{validation set}, we tried $\sigma^2 = \{0.01, 0.1, 1, 10\}$. The best performing value on the validation set is then chosen.


Getting the code of each background pixel
Decomposing the image into 9 layers
Layer # :
1
2
3
4
5
6
7
8
9

0	7	0	0	0	0	6	6	6	6	6	6	6	6	6	6	6	0	6	0
0	0	0	4	4	4	6	6	6	6	6	6	6	6	6	6	6	0	6	0
0	0	4	4	4	4	6	6	6	6	6	6	6	6	6	6	6	0	6	0
0	0	0	4	4	4	6	6	6	6	6	6	6	6	6	6	6	0	6	0
0	9	0	0	4	4	6	6	6	6	6	6	6	6	6	6	6	0	6	0
0	9	9	9	0	0	6	6	6	6	6	6	6	6	6	6	6	0	6	0
0	9	9	9	9	9	0	0	0	6	6	6	6	6	6	6	6	0	6	0
0	9	9	9	9	9	5	5	5	0	0	6	6	6	6	6	6	0	6	0
0	9	9	9	9	9	5	5	5	0	0	6	6	6	6	6	6	0	6	0
0	9	9	9	9	9	5	0	0	0	4	6	6	6	6	6	6	0	6	0
0	9	9	9	9	9	0	0	4	4	4	6	6	6	6	6	6	0	6	0
0	9	9	9	9	9	0	4	4	4	4	6	6	6	6	6	6	0	6	0
0	9	9	9	9	9	0	4	4	4	4	6	6	6	6	6	6	0	6	0
0	9	9	9	9	9	0	0	4	4	4	6	6	6	6	6	6	0	6	0
0	9	9	9	9	9	9	0	0	0	4	6	6	6	6	6	6	0	6	0
0	9	9	9	9	9	9	9	9	0	0	0	6	6	6	6	6	0	6	0

Fig. 9 Illustrating how local directional features are formed (refer to Sect. 4.6). Note that layer # i shows only background pixels having code # i

(v) *The dimensionality after applying PCA for the Gaussian classifier* (d').
we try $d' = \{d, 0.9d, 0.8d, \ldots\}$ till we reach a feature vector of length 5, where d is the original dimensionality of the feature vector. The best performing value on the validation set is then chosen.

Table 3 shows the parameters values used for different combinations of classifiers and features. Table 4 shows the accuracy of each combination on the MADBase test set. Table 5 shows the *classification* timing performance (not including feature extraction time) of each of the features/classification. We measured the timing performance for a certain classifier/features combination by the ratio of the CPU seconds needed to classify one pattern and the CPU seconds needed to classify one pattern using the classifier/features that have the least timing. The classifier/features combination that gives least timing was found to be linear SVM /Wavelet.

The column named "average accuracy" of Table 4 shows the average accuracy of each feature set. This is used as an assessment of the powerfulness of each feature set. The last column orders the feature sets according to their average accuracies. Similarly, the row named "average accuracy" shows the average accuracy of each classifier as an assessment of classification techniques powerfulness. The last row

Table 3 Classifiers parameters settings

	Parzen	2-NN	Linear SVM	RBF SVM		GC
	σ^2	h	C	C	γ	d'
Gradient	0.1	50	0.1	100	0.1	60
Kirsch	0.1	200	1	100	0.01	50
Local chain	0.1	50	1	50	0.1	60
Wavelet	1	150	0.1	10	0.03	26
Low dim. conc.	1	250	0.01	30	0.002	141
Local directional	0.01	100	10	10	0.3	45

of the table orders the classifiers according to their average accuracies.

5.1 The problem of the Arabic digit ‘0’

The Arabic digit ‘0’ is just a dot which can be of various peculiar shapes when written fast by the hand as shown in Fig. 10. Digit ‘0’ may get confused with almost all other digits (especially 1 and 5). However, the eye can easily differentiate between them because the digit ‘0’ is clearly much smaller than any other digit. Table 6 shows the confusion matrix of the most powerful classifier/features pair: RBF SVM/gradient. It is clear from this confusion matrix that most of errors involve the digits ‘0’. One way to solve this problem is to introduce a size sensitive feature to all the feature sets we used (this means that the length the feature vector of the gradient features becomes 201, and that of Kirsch becomes 101, etc.). This helps greatly in reducing the confusion the digit ‘0’ introduces. The size-sensitive feature we propose is the area of the digit bounding box to the average bounding box areas of the training set digits. Because the MADBase digits are size-normalized, we calculated the size-sensitive feature using the original digit images of ADBase (see Sect. 2 for a discussion of ADBase and MADBase). Table 7 shows the confusion matrix of RBF SVM/gradient after adding the size-sensitive feature. Table 8 displays the accuracies of different classifier/features pairs on MADBase after adding the size-sensitive feature extracted from ADBase.

5.2 Latin digits

For the sake of comparison, we applied the same classifiers/features combinations to Latin digits of the MNIST library.

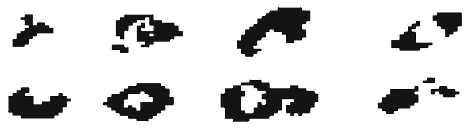


Fig. 10 Some samples of the digit ‘0’ from ADBase

Table 4 Accuracies of classifier/features pairs on MADBase

	KNN	Parzen	2-NN	1-NN	PCA + Quad	Linear SVM	RBF SVM	GC	Fisher	Avg. accuracy	Rank
Gradient	98.9	98.92	98.98	99.02	99.11	99.03	99.18	97.94	98.27	98.8167	1
Kirsch	98.14	98.02	98.5	97.96	98.37	98.82	98.87	97.25	97.57	98.1556	4
Local chain	95.41	94.32	96.12	95.97	96.46	96	97.08	93.01	95.72	95.5967	6
Wavelet	98.56	98.34	98.43	96.54	98.03	97.2	98.85	96.23	95.53	97.5267	5
Low dim. conc.	98.1	98.19	98.77	98.51	98.25	98.9	98.79	97.54	98.49	98.393	3
Local directional	98.68	98.13	98.57	98.8	98.82	98.73	99.14	96.79	98.73	98.4878	2
Avg. accuracy	97.97	97.65	98.233	97.8	98.232	98.11	98.65	96.51	97.385		
Rank	5	7	2	6	3	4	1	9	8		

Table 5 Classification times of classifier/features pairs

	KNN	Parzen	2-NN	1-NN	PCA + Quad	Linear SVM	RBF SVM	GC	Fisher	Avg. complexity	Rank
Gradient	3403	4215	3	3	8	3	495	13	3	905	4
Kirsch	1890	3273	7	3	6	1	230	5	1	602	2
Local chain	3403	4215	3	3	8	3	803	13	3	939	5
Wavelet	1333	2128	4	2	5	1	445	3	1	436	1
Low dim. conc.	2760	4523	11	3	7	2	295	48	2	850	3
Local directional	3835	5218	6	4	8	3	478	10	3	1063	6
Avg. complexity	2771	3929	6	3	7	2	458	15	2		
Rank	7	8	3	2	4	1	6	5	1		

Table 6 Confusion matrix of RBF SVM/gradient without using the size-sensitive feature

Input digit	Recognized as 0	1	2	3	4	5	6	7	8	9
0	983	6	0	0	0	11	0	0	0	0
1	13	982	1	1	1	0	0	0	1	1
2	2	0	994	1	2	1	0	0	0	0
3	0	0	5	994	1	0	0	0	0	0
4	0	2	1	0	996	0	0	0	0	1
5	5	0	1	0	0	991	0	0	1	2
6	0	3	0	0	2	0	993	0	0	2
7	0	1	0	0	0	1	2	996	0	0
8	0	0	1	0	0	0	1	0	995	3
9	0	1	1	0	2	0	2	0	0	994

Table 7 Confusion matrix of RBF SVM/gradient when using the size-sensitive feature

Input digit	Recognized as 0	1	2	3	4	5	6	7	8	9
0	989	3	0	0	0	8	0	0	0	0
1	2	994	1	1	1	0	0	0	0	1
2	1	1	994	1	2	1	0	0	0	0
3	0	0	4	996	0	0	0	0	0	0
4	0	1	1	0	997	0	0	0	0	1
5	2	0	1	0	1	994	0	0	1	1
6	0	1	0	0	2	0	996	0	0	1
7	0	0	0	0	0	1	2	997	0	0
8	0	0	1	0	0	0	0	0	996	3
9	0	0	1	0	2	0	2	0	0	995

Table 8 Accuracies of classifier/features pairs on MADBase after adding the size-sensitive feature extracted from ADBase

	KNN	Parzen	2-NN	1-NN	PCA + Quad	Linear SVM	RBF SVM	GC	Fisher	Avg. accuracy	Rank
Gradient	99.03	98.86	99.2	99.18	99.28	99.22	99.48	98.2	98.72	99.019	1
Kirsch	98.27	98.15	98.57	98.29	98.73	99.06	99.13	97.84	98.06	98.456	3
Local chain	97.91	97.66	98.15	98.28	98.5	98.32	98.62	94.9	97.47	97.757	6
Wavelet	98.57	98.46	98.82	97.33	98.14	97.73	99.05	96.41	96.53	97.894	5
Low dim. conc.	98.11	98.19	98.77	98.72	98.75	98.9	98.79	97.79	98.68	98.463	4
Local directional	98.89	97.97	99.04	99.18	99.18	99.1	99.29	97.16	98.28	98.677	2
Avg. accuracy	98.46	98.215	98.76	98.5	98.76	98.72	99.06	97.05	97.957		
Rank	6	7	3	5	2	4	1	9	8		

The results are summarized in Table 9. We see that the best classifier/features combination for both Arabic and Latin digits is RBF SVM with gradient features and both gain nearly the same accuracy (around 99.4%). This may lead to the conclusion that the two problems are the same. However, if we look at the less powerful classifiers/features combinations in Tables 8 and 9, we see obviously that the results of Latin and Arabic digits are different. The reason why RBF SVM and gradient features always give the best results is that they are both powerful, not because different

Table 9 Accuracies of classifier/features pairs on Latin digits of MNIST

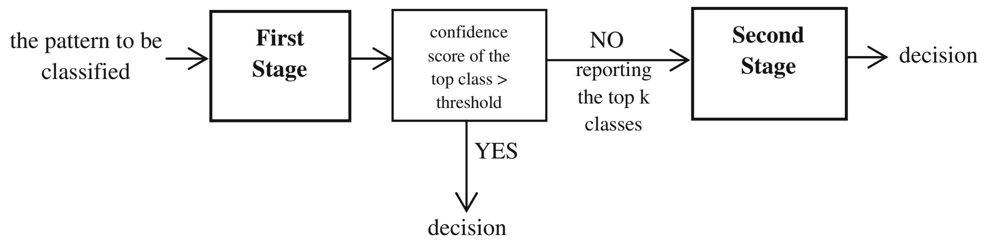
	KNN	Parzen	2-NN	1-NN	PCA + Quad	Linear SVM	RBF SVM	GC	Fisher	Avg. accuracy	Rank
Gradient	98.74	98.78	98.97	98.73	99.09	98.82	99.44	97.99	98.25	98.757	1
Kirsch	96.08	95.86	97.26	96.45	97.83	97.95	98.67	97.17	97.21	97.16	4
Local chain	95.46	93.81	97.46	96.95	97.23	97.14	98	95.1	95.72	96.319	5
Wavelet	96.78	96.75	98.01	93.95	96.61	94.46	98.44	95.53	92.74	95.91889	6
Low dim. conc.	96.83	96.16	98.19	97.38	97.18	97.67	98.6	96.29	97.21	97.279	2
Local directional	96.6	95.9	98.09	97.77	97.91	97.91	98.5	95.57	96.71	97.2178	3
Avg. accuracy	96.75	96.21	98	96.87	97.64	97.33	98.61	96.275	96.3067		
Rank	6	9	2	5	3	4	1	8	7		

techniques perform the same for Arabic and Latin digits. But one may ask: what is the importance of the fact that less powerful classifiers performs differently on Arabic and Latin digits while the best classification technique performs equally likely? At the end, we will use the most powerful technique. This is really true if we used one stage classifier. But if we wanted to use a two-stage classifier, the less powerful classifiers will be used; and hence, the difference between Arabic and Latin digits will lead to different structures of the two-stage system. This will be made clear in Sect. 6.

6 An optimal two-stage classification system

As noticed from Table 8, the best accuracy on MADBase is achieved by RBF SVM with gradient features (99.48%). However, the classification time of the SVM is very high making the system not practical. A well-known technique to speed up the classification process is to use a two-stage structure in which a fast classifier is put as a first stage and a time consuming classifier, yet more powerful than the first stage, is put as a second stage [27,28]. The pattern to be classified is handled first by the first stage. If the classification decision of the first stage is made with high confidence score (the confidence score of the top class is higher than a predefined threshold), its decision is accepted and the classification process ends. If the decision confidence is below that threshold, it is rejected to be classified by the second stage. An improvement of this procedure is possible if the second stage has a different module for each class. In this case, when the first stage rejects some pattern to the second stage, it indicates also which classes had the k highest confidence scores [8]. Hence, the second stage could safely evoke the modules responsible for only those *k* classes saving considerable processing time. Figure 11 illustrates this idea.

In this section, we are going to search for the optimal two-stage system for the Arabic digit recognition problem using the classification and feature extraction techniques reported in this study. We mean by 'optimal two-stage system' the one that achieves the lowest recognition time under the condition that it achieves the highest possible accuracy. We have 54 different features/classifier combinations; this means that there are 54×53 possible two-stage systems. To find the optimal two-stage system, we may try all the 54×53 possible two-stage structures, and then selects the optimal one. However, this encounters unnecessary trials. Our definition of 'optimality' asserts that the two-stage system should achieve the highest possible accuracy. This could be achieved if we put the classifier that achieves the highest possible accuracy as second stage. It is true that a two-stage system could achieve accuracy higher than the second stage alone, but the increase in the accuracy, if it happens at all, is usually negligibly small. This reasoning leads us to put the RBF SVM with gradient features as the second stage which reduces the search space to only 53 trials. The search space could be reduced further if we study the case when a classifier with a feature extraction technique other than the gradient features is used as a first stage. In this case the feature extraction step is evoked twice, one for the first stage and the other for the second stage. On the other hand, if we used the gradient features with the first stage, the feature extraction step is executed only once to generate a feature vector for both first and second stage. Moreover, the gradient features set gains the highest accuracy with almost all classifiers as obvious from Table 8; this means that it will be the best choice for all first stage candidates. Thus we may safely not consider the features/classifier combinations that do not use the gradient features. Now, we are left with only 9 trials. If we note that the KNN and Parzen classifier are very time consuming and have recognition time more than the second stage itself, we can see obviously that they could not be possible candidates to be first stage. Finally, the Fisher classifier has the same timing performance as the linear SVM but with lower accuracy which means that the linear SVM is for sure a better candidate to be first stage than the Fisher classifier. The same concept applies for the Gaussian classifier as it has higher recognition time than the linear SVM but with lower accuracy. Now we are left with four possible candidates for



the pattern to be classified
First Stage
confidence score of the top class > threshold
NO reporting the top k classes
Second Stage
decision
YES
decision

Fig. 11 Structure of the proposed two-stage classification system

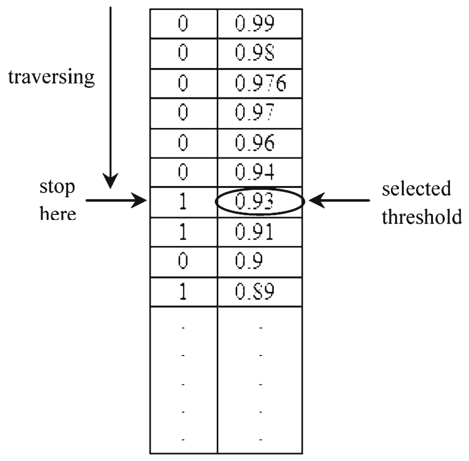
the first stage: (i) 1-NN, (ii) linear SVM, (iii) 2-NN, and (iv) PCA + Quadratic; all with gradient features. Our strategy is then to try the mentioned four classification techniques with gradient features as first stage and RBF SVM with gradient features as second stage on the test set and pick the one that achieves the lowest recognition time provided that the overall accuracy is as high as that of the second stage.

Now we are left with the problem of deciding the value of the first stage threshold and the number of classes k that are passed to the second stage. In the following two subsections, strategies for deciding the threshold and the parameter k are going to be introduced.

6.1 First stage threshold

As our goal is to reduce the recognition time while retaining the accuracy of the second stage, the first stage threshold may be selected to be the one that commits no errors. This prevents the error leakage from the first stage. However, this will be too strict a strategy. We actually can allow the first stage to commit some errors if these errors are also committed by the second stage. Hence, our strategy of picking the first stage threshold might be as follows. We first train the first stage on the set {training set}-{validation set} (see Sect. 5 for how we chose the validation set), then use it to classify the patterns of the validation set. We store a record for each classified pattern of the validation set. Each record contains two entries: (i) whether the pattern is wrongly classified while correctly classified by the second stage (given code ‘1’) or otherwise (given code ‘0’) and (ii) the corresponding decision score of the top class. And then all the validation set patterns records are ordered from high to low according to their decision score. This ordered list is then traversed form top to bottom searching for the first record of pattern that is wrongly classified and correctly classified by the second stage (i.e., marked with code ‘1’). When we find such pattern, we stop traversing the list and declare the decision score of this pattern to be the first stage threshold. This process is illustrated with a hypothetical example in Fig. 12.

We add a note here that 1-NN and linear SVM were designed using the OVO scheme which does not provide a smooth confidence score which is necessary for acceptable rejection performance. To alleviate this problem, we used the technique devised by Price et al. [29] to produce an output probability of the OVO scheme using the output probabilities



0	0.99
0	0.98
0	0.976
0	0.97
0	0.96
0	0.94
1	0.93
1	0.91
0	0.9
1	0.89
.	.
.	.
.	.
.	.
.	.

Fig. 12 A hypothetical example illustrating first stage threshold selection process. Each *row* is a record of a different pattern of the validation set. The *right entry* of each record is labels ‘1’ if a classification error is committed while the pattern is correctly classified by the second stage (RBF SVM with gradient features). The *second entry* of each record is the top decision score of the corresponding pattern. The records are ordered according to the decision score entry

of all the binary classifiers of the scheme. This worked well with 1-NN, but not with linear SVM as the binary linear SVM does not even produce probabilistic output. The linear SVM was made to give probabilistic output using the calibrating procedure devised by Platt [30].

6.2 The parameter k

The process by which we pick a value of k is described as follows. For each value of k (which ranges from 1 to 10), we calculate the number of errors committed. We consider an error is committed if the true class of the pattern is not among the top k classes. The value of k that leads to zero errors (in the sense described) is chosen. However, this will be too strict a criterion for choosing k as we can be tolerant for some errors to be committed as far as they are also committed by the second stage. Hence, we choose the value of k that leads to zero errors in the first stage that are correctly classified by the second stage.

We note here that the RBF SVM (the second stage) actually does not have a different module for each class, but a different module for each pair of classes. However, this is not a problem as we can tailor a sub-classifier for the top-k classes

Table 10 The performance of different first stage classifier candidates for MADBase

	2-NN	1-NN	PCA + Quad	Linear SVM
k	4	3	6	6
Number of pattern rejected to reach 1 error (out of 10,000)	290	1095	573	439
Accuracy without first stage	99.48	99.48	99.48	99.48
Accuracy with first stage	99.49	99.47	99.48	99.48
Recognition time without first stage	512.5	512.5	512.5	512.5
Recognition time with first stage	20	22.5	42.5	27.5
Rank	1	2	4	3

Table 11 The performance of different first stage classifier candidates for MNIST

	2-NN	1-NN	PCA + Quad	Linear SVM
k	4	4	4	5
Number of pattern rejected to reach 1 error (out of 10,000)	815	1436	545	1435
Accuracy without first stage	99.44	99.44	99.44	99.44
Accuracy with first stage	99.41	99.44	99.41	99.45
Recognition time without first stage	657.5	657.5	657.5	657.5
Recognition time with first stage	35	32.5	27.5	42.5
Rank	3	2	1	4

using the binary classifiers concerning each of the top-k classes only.

6.3 Selecting the optimal cascade

Table 10 shows the k values to reach zero error and number of patterns rejected to reach 1 error pattern for the first stage classifiers candidates. These two quantities are calculated using the validation set. Table 10 also shows the accuracy and recognition time of the overall system on the test set if each of the classifiers candidates is used as a first stage with k values and thresholds calculated using the validation set. The table indicates that 2-NN is the best candidate to be a first stage. Note here that the feature vector used is the one that includes the size-sensitive feature. This is why the timing performance of second stage alone (the RBF SVM with gradient features) appears in Table 10 is different from that appears in Table 5 in which the original feature vector without the size-sensitive feature is used.

For the sake of comparison, we applied the same procedure for selecting a fast two-stage classification system for Latin digits. The results are shown in Table 11. We note here that the selection of the best two-stage system is not obvious whether it should be 1-NN or PCA-Quad as one achieves higher accuracy than the other but with higher recognition time. But it is clear that 2-NN (which is the best choice for the case of Arabic digits) is not the best choice for Latin digits. This means that while the best individual classifier for both Arabic and Latin digits is the same (RBF SVM with gradient features) the optimal two-stage system for them is not the same. This means that careful study of the problem at hand is necessary even if it looks very close to other known problems.

7 Conclusion

In this paper, we have evaluated the performance of a number of feature sets and classification techniques on the problem of recognizing Arabic digits. Since there is very little research that has been done on Arabic digits so far, the results of our work hopefully serves as a benchmark for future research on Arabic digits. In addition, we have introduced a new feature extraction technique: the “local directional features”. Out of the results we identified and alleviated the problem introduced by the digit ‘0’. Moreover, we proposed a strategy to select and design an optimal two-stage system and used it to suggest a two-stage system for Arabic digits that achieved very high accuracy with low recognition time.

References

1. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
2. Dong, J.: Comparison of algorithms for handwritten numeral recognition. Technical report, CENPARMI, Concordia University, 1999

3. Liu, C.-L., Nakashima, K., Sako, H., Fujisawa, H.: Handwritten digit recognition: benchmarking of state-of-the-art techniques. Pattern Recognit. **36**, 2271–2285 (2003)
4. Zhang, P., Bui, T., Suen, C.Y.: Hybrid feature extraction and feature selection for improving recognition accuracy of handwritten numerals. Proc. 8th ICDAR, pp. 136–140, 2005
5. Teow, L., Loe, K.: Robust vision-based features and classification schemes for off-line handwritten digit recognition. Pattern Recognit. **40**(6), 1816–1824 (2007)
6. Belongie, S., Malik, J., Puzicha, J.: Shape matching and object recognition using shape contexts. PAMI, vol. 24, no. 24, 2002
7. Lauera, F., Suen, C., Blocha, G.: A trainable feature extractor for handwritten digit recognition. Pattern Recognit. **36**, 2271–2285 (2003)
8. Gorgevik, D., Cakmakov, D.: An efficient three-stage classifier for handwritten digit recognition. Proc. 17th ICPR, pp. 1051–4651 (2004)
9. Al-Omari, F., Al-Jarrah, O.: Handwritten Indian numerals recognition system using probabilistic neural networks. Adv. Eng. Inform. **18**(1), 9–16 (2004)
10. Said, F., Yacoub, R., Suen, C.: Recognition of English and Arabic numerals using a dynamic number of hidden neurons. Proc. 5th ICDAR, pp. 237–240, 1999
11. El-Sherif, E., Abdelazeem, S.: A two-stage system for Arabic handwritten digit recognition tested on a new large database. International Conference on Artificial Intelligence and Pattern Recognition (AIPR-07), Orlando, FL, USA, July 2007, pp. 237–242
12. Mozaffari, S., Faez, K., Ziaratban, M.: Structural decomposition and statistical description of Farsi/Arabic handwritten numeric characters. Proc. 8th ICDAR, pp. 237–241, 2005
13. Soltanzadeh, H., Rahmati, M.: Recognition of Persian handwritten digits using image profiles of multiple orientations. Pattern Recognit. Lett. **25**(14), 1569–1576 (2004)
14. Duda, R., Hart, P., Strok, D.: Pattern Recognition, 2nd edn. Wiley, New York (2000)
15. Webb, A.: Staistical Pattern Recognition, 2nd edn. Wiley, New York (2002)
16. Furnkranz, J.: Round robin classification. J. Mach. Learn. Res., vol. 2, pp. 721–747, 2002
17. Friedman, J.: Another approach to polychotomous classification. Technical Report, Stanford University, 1996
18. Bennett, K.P., Mangasarian, O.L.: Multicategory discrimination via linear programming. Optim. Methods Softw. **3**, 27–39 (1994)
19. Cortes, C., Vapnik, V.: Support vector networks. Mach. Learn. **20**(3), 273–297 (1995)
20. Gonzales, R.C., Woods, R.E.: Digital image processing, 2nd edn. Addison-Wesley, Reading, MA, USA (2002)
21. Zhang, D., Lu, G.: Review of shape representation and description techniques. Pattern Recognit., vol. 37, 2004
22. Zhang, P., Bui, T., Suen, C.: Hybrid feature extraction and feature selection for improving recognition accuracy of handwritten numerals. Proc. 8th ICDAR, 2005
23. Trier, O.D., Jain, A.K., Taxt, T.: Feature extraction methods for character recognition—a survey. Pattern Recognit. **29**(4), 641–662 (1996)
24. Hanmandlu, M., Kumar, H., Mohan, K.: Neural based handwritten character Recognition. Proc 5th, ICDAR, p. 241, 1999
25. de Oliveira, J., de Carvalho, J., Freitas, C., Sabourin, R.: Feature sets evaluation for handwritten word recognition using a baseline system. Proc. 8th IWFHR, pp. 446–451, 2002
26. Hsu, C., Lin, C.: A comparison of methods for multi-class support vector machines. IEEE Trans. Neural Netw. **1**(13), 415–425 (2002)
27. Kaynak, C., Alpaydin, E.: MultiStage cascading of multiple classifiers: one man's noise is another man's data. Proc. 17th ICML, pp. 455–462, (2000)
28. Alpaydin, E., Kaynak, C., Alimoglu, F.: Cascading multiple classifiers and representations for optical and pen-based handwritten digit recognition. Proc. 7th IWFHR, pp. 453–462, 2000
29. Price, D., Knerr, S., Personnaz, L., Dreyfus, G.: Pairwise neural network classifiers with probabilistic outputs. NIPS, pp. 1109–1116, 1995
30. Platt, J.: Probabilistic outputs for support vector machines and comparison to regularized likelihood methods. In: Smola, A., Bartlett, P., Scholkopf, B., Schuurmans, D. (eds.) Advances in Large Margin Classifiers. Cambridge, MA, pp. 61–74, 2000
31. Davies, E.R.: Machine Vision: Theory, Algorithms, Practicalities. Morgan Kaufmann Publishers Inc., San Francisco, CA (2004)