A Multiple Feature/Resolution Approach to Handprinted Digit and Character Recognition

John T. Favata and Geetha Srikantan

CEDAR, State University of New York at Buffalo, Buffalo, NY 14260

ABSTRACT

This article outlines the philosophy, design, and implementation of the Gradient, Structural, Concavity (GSC) recognition algorithm, which has been used successfully in several document reading applications. The GSC algorithm takes a quasi-multiresolution approach to feature generation; that is, several distinct feature types are applied at different scales in the image. These computed features measure the image characteristics at local, intermediate, and large scales. The local-scale features measure edge curvature in a neighborhood of a pixel, the intermediate features measure short stroke types which span several pixels, and the large features measure certain concavities which can span across the image. This philosophy, when coupled with the *k*-nearest neighbor classification paradigm, results in a recognizer which has both high accuracy and reliable confidence behavior. The confidences computed by this algorithm are generally high for valid class objects and low for nonclass objects. This allows it to be used in document reading algorithms which search for digit or character strings embedded in a field of objects. Applications of this paradigm to off-line digit string recognition and handwritten word recognition are discussed. Tests of the GSC classifier on large data bases of digits and characters are reported.

I. INTRODUCTION

Many different approaches have been used by researchers to solve the problem of computer handwritten digit and character recognition [1]. These approaches have included investigations of: feature sets [2,3], classifier algorithms multiple combinations of classifiers [4], and novel statistical methods [5]. There can be much overlap between different methods, and a precise taxonomy can prove difficult to formulate. It would be safe to say that the precise algorithm classification has much to do with the investigator's perspective during the design of the algorithm.

Many different algorithms have been explored by the researchers at CEDAR [6,7]. These algorithms have encompassed a wide range of feature and classifier types. Each algorithm has characteristics, such as high speed, high accuracy, good thresholding ability, and generalization, which are useful for specific applications. This article will outline the philosophical and practical details of one of these classifiers: the Gradient, Structural, Concavity (GSC) classifier.

II. PHILOSOPHY

The approach used in designing the GSC features was based on the observation that feature sets can be designed to extract certain types of information from the image. Feature detectors can be built to detect the local, intermediate, and global features of an image. The basic unit of a digitized image is the pixel with the location (x,y coordinate) and a relationship to its neighbors at different ranges from locally to globally. This can be expressed by saying that we want to determine the relationship of each pixel to every other pixel at increasing distances. In a sense, this is taking a multiresolution approach to feature generation. The GSC features approximate a heterogeneous multiresolution paradigm by being generated at three ranges: local, intermediate, and global.

The gradient features detect local features of the image and provide a great deal of information about stroke shape on a small scale. The structural features extend the gradient features to longer distances and give useful information about stroke trajectories. The concavity features are used to detect stroke relationships at long distances which can span across the image. In practice, there are computationally imposed limits to how a particular philosophy can be implemented. In the GSC algorithm, decisions were made in the exact detection and representation of the features to result in a practical algorithm. The exact implementation should not distract from the underlying philosophy. It should be emphasized that this article presents one particular implementation of the GSC philosophy, and that others are possible. The total feature vector length for this implementation is 512 bits. The GSC feature vector is very compact because it is binary; other algorithms may use a smaller number of multivalued (or real) features, but the effective number of bits to represent such feature vectors can actually be quite large.

III. FEATURE DESCRIPTION

The GSC algorithm was designed to work with binarized images, so it is presumed that the image has been thresholded using a suitable algorithm [8]. The image is slant normalized using a moment-based algorithm to reduce the effects of skew. A bounding box is placed around the image and the features are computed (Fig. 1). The feature maps are sampled by placing 4×4 grid on the maps (see Fig. 5a) and computing the features present in each region. The features themselves are computed independently of this sampling grid.

A. Gradient Features. The gradient features are computed by convolving two 3×3 Sobel operators on the binary image. These operators approximate the x and y derivatives in the image at a pixel position (Fig. 2). The gradient of a center pixel is computed

Revised manuscript received 29 May 1996

CCC 0899-9457/96/040304-08

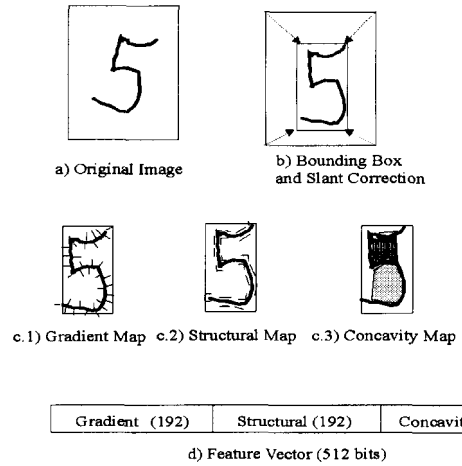

Figure 1. General flow of GSC feature generation.

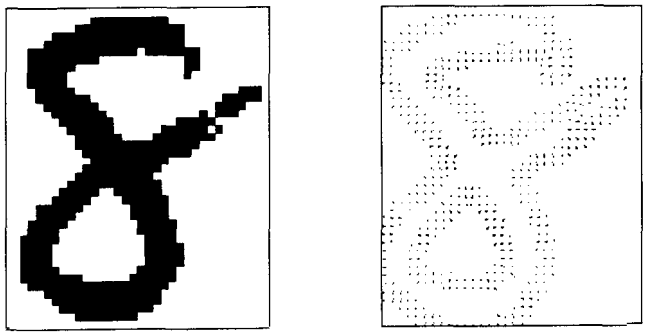

Figure 2. Gradient map of digit 8.

as a function of its eight nearest neighbors (Fig. 3a). The vector addition of the operators' output is used to compute the gradient of the image. Since the gradient is vector valued with magnitude and direction, only the direction is used in the computation of a feature vector. The direction of the gradient can range from 0.0 to 2π radians. This range is split into 12 nonoverlapping regions of $2\pi/12$ radians (Fig. 3b). In each sampling region (4×4 grid), a histogram is taken of each gradient direction at each pixel which lies in the region. A threshold is applied to the histogram and the corresponding feature bit is set for each feature count that exceeds the threshold. This subset of the GSC features produces $12 * 4 * 4 = 192$ bits of the total feature vector.

There are many ways to approximate the gradient of a two-dimensional image. The 3×3 Sobel operator was chosen because of its computational simplicity and useful approximation of the gradient for this problem, and early tests indicated that it performed better than other approximations. It is emphasized that this presentation of the gradient features represents only one possible solution in the spectrum of solutions for finding local features of an object. The designer of the algorithm should pick the operator(s) most appropriate to the problem at hand.

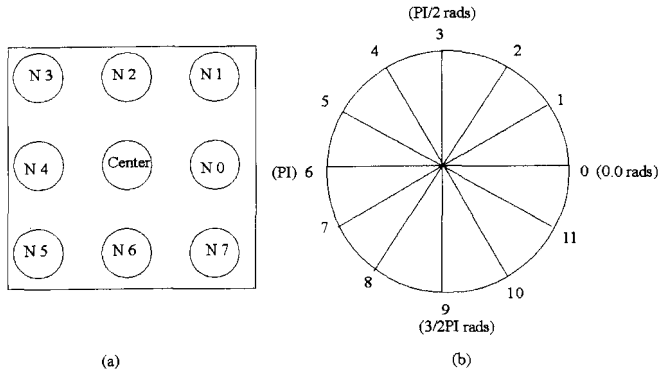

Figure 3. (a) Eight connectivity definitions; (b) 12 equispaced sampling regions.

B. Structural Features. The structural features capture certain patterns embedded in the gradient map. These patterns are "mini-strokes" of the image. A set of 12 rules is applied to each pixel. These rules operate on the eight nearest neighbors of the pixel. Each rule examines a particular pattern of neighboring pixels for allowed gradient ranges. For example, rule 1 states that if neighbor (N 0) of a pixel (see Fig. 3 for connectivity and gradient definitions) has a gradient range of 2,3,4 and neighbor (N 4) has gradient range 2,3,4, then the rule is satisfied and its corresponding bit in the feature vector is set to 1. The current rule set, listed in Table I, includes four types of corners and eight types of lines. These rules were derived from the gradient features for computational simplicity and are correlated to them. However, other implementations of GSC can use structural features derived from different operators and employ different rules for extraction. These features contribute $4 \times 4 \times 12 = 192$ bits to the total feature vector.

C. Concavity Features. These features, which are the coarsest of the GSC set, can be broken down into three subclasses of features: coarse density, large stroke, and concavity. The total contribution of these features are $4 \times 4 \times 8 = 128$ bits.

1. Coarse Pixel Density Features. These features capture the general groupings of pixels in the image. They are computed by placing the 4×4 sampling grid on the image and counting the number of image pixels that fall into each grid. Thresholding converts these area counts into a single bit for each region. This feature contributes $4 * 4 = 16$ bits of the feature vector.

2. Large-Stroke Features. These features attempt to capture large horizontal and vertical strokes in the image. Run lengths of horizontal and vertical black pixels across the image are first computed. From this information, the presence of strokes are

Table I. Structural feature definitions.

Rule	Description	Neighbor 1 (Range)	Neighbor 2 (Range)
1	Horizontal line, type 1	N 0 (2,3,4)	N 4 (2,3,4)
2	Horizontal line, type 2	N 0 (8,9,10)	N 4 (8,9,10)
3	Vertical line, type 1	N 2 (5,6,7)	N 6 (5,6,7)
4	Vertical line, type 2	N 2 (1,0,11)	N 6 (1,0,11)
5	Diagonal rising, type 1	N 5 (4,5,6)	N 1 (4,5,6)
6	Diagonal rising, type 2	N 5 (10,11,0)	N 1 (10,11,0)
7	Diagonal falling, type 1	N 3 (1,2,3)	N 7 (1,2,3)
8	Diagonal falling, type 2	N 3 (7,8,9)	N 7 (7,8,9)
9	Corner 1	N 2 (5,6,7)	N 0 (8,9,10)
10	Corner 2	N 6 (5,6,7)	N 0 (2,3,4)
11	Corner 3	N 4 (8,9,10)	N 2 (1,0,11)
12	Corner 4	N 6 (1,0,11)	N 4 (2,3,4)

determined by testing for stroke lengths above a threshold. This feature contributes $4*4*2 = 32$ bits of the feature vector.

3. U/D/L/R/H Concavity Features. These features are computed by convolving the image with a starlike operator. This operator shoots rays in eight directions and determines what each ray hits. A ray can hit an image pixel or the edge of the image. A table is built for the termination status of the rays emitted from each white pixel of the image. A computationally efficient algorithm similar to run-length encoding is actually used to compute the star operator. The class of each pixel is determined by applying rules to the termination status patterns of the pixel. Currently, upward/downward, left/right pointing concavities are detected along with holes. The rules are relaxed to allow nearly enclosed holes (broken holes) to be detected as holes. This gives a bit more robustness to noisy images. These features can overlap in that in certain cases more than one feature can be detected at a pixel location. These features contribute $4 \times 4 \times 5 = 80$ bits.

IV. CLASSIFICATION

The classification problem can be stated as finding functions which map feature vectors to classes. Ideally these functions should map hyperspheres (clusters of same class objects) in the feature space to the class space. With high-dimensional feature spaces, this can be a very difficult problem. Depending on the sensitivity of the parameters in the classification function, there may never be enough training data to estimate these functions adequately in certain regions of the feature space. In addition, it is also assumed that our features may not have enough resolving power to distinguish among certain cases of images or there may be an aliasing (many-to-one) problem where two different image patterns map into the (nearly) same feature vector. This can be due to practical implementation compromises involved in the algorithm design such as the need for speed or machine memory size limitations. In the domain of handwritten images it is also possible that certain cases of digits or characters may be so borderline or poorly written that even humans could misrecognize them (Fig. 4).

In computing the classification functions, it is necessary to find functions that accurately reflect the training set and generalize well to the testing sets. This means that the functions should accurately recognize members of the training set with high confidence and smoothly roll off as the feature vectors move away from the labeled vectors of the training set. That is, as the underlying image is smoothly transformed into another image of a different class, there should be a smooth transition of the classification function. This property is useful to prevent spurious responses in those regions of the feature space which are inadequately represented in the training set. In addition, the classifier must have good behavior in invalid regions of the space. These regions represent feature vectors which do not belong to class objects (the reject class). Vectors which are mapped into these regions should produce minimal confidences. This last property is important if the recognizer is used to distinguish between valid and invalid objects such as using the algorithm to find valid digits embedded in a field of unlabeled objects. This work has focused on the weighted k-nearest neighbor classifier, although others have tried with various degrees of success.

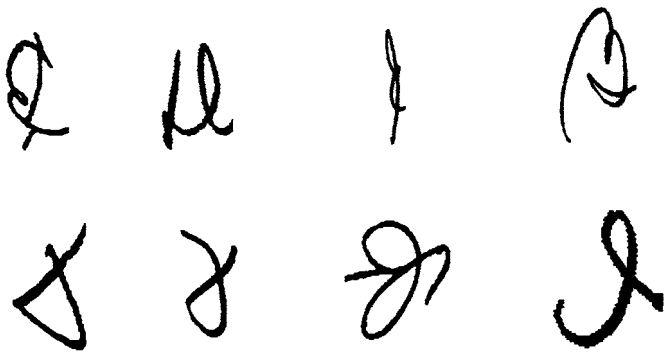

Figure 4. Ambiguous characters.

The k-nearest neighbor (k-nn) [9] approach attempts to compute a classification function by using the labeled training points as nodes or anchor points in the n ($n = 512$) dimensional space. In a sense, this is the most detailed description of the space that is possible from the training samples assuming that the form of the probability distribution function (PDF) is unknown (nonparametric PDF) or has a number of parameters that are very difficult to approximate. Rather than using a 1-nearest neighbor classifier, a k-nn classifier is used to reduce the effect of mislabeled training data and to get a better estimate of the class PDF at any particular point in the space. By choosing an appropriate distance metric as a weighting function, a smooth roll off in response can be obtained as a feature vector is moved away from a cluster center. Since the feature vector is binary (a bit vector), the comparison between unknown and labeled vectors involves bit operations which can be done in parallel at a machine's integer word length (with logical AND).

The main disadvantage of k-nn is the memory required to hold the training data and the classification speed. Two techniques have been developed to speed classification with negligible degradation in performance: prototype thinning and clustering. Prototype thinning reduces the overall density of the prototypes in feature space by finding those prototypes which are redundant. This is done by finding prototypes which are strictly surrounded by prototypes of the same class. These prototypes can be safely removed while those that are close to different classes (boundary points) are kept. This technique is most successful when there are many training prototypes for each class. When the class training set is sparse, clustering techniques have been to greatly reduce the number of comparisons necessary to implement k-nn. The idea is to find a number of prototypes which can serve as centers of hyperspheres of varying diameter in the feature space. Each center contains a number of prototypes of any class (its neighbors) within a radius r. These centers and their associated neighbors are stored in a data structure. Classification is done by comparing the unknown prototype to each of the centers and then picking the N closest centers and their associated neighbors as the prototypes which are most likely candidates for full k-nn classification. This technique can significantly reduce the number of comparisons necessary, since only those closest prototypes to the unknown are used for classification.

This work does not attempt to solve the classic problems of classification, because they are highly dependent on the definition of the problem. The solution presented here is specific to the binary feature space, the GSC feature set, and the type of training or testing objects used. There also exist other solutions to the prototype storage problem which depend on the distribution of feature vectors in the space. These distributions are a function of the type of feature space, the feature set, and the number of object classes.

A. Distance Metrics. There are a number of useful distance metrics for comparing two bit vectors $\mathbf{V}_1$ and $\mathbf{V}_2$ [10]. If we define (see [10] for definition details) n_{00} as the number of 0 bits that match between $\mathbf{V}_1$ and $\mathbf{V}_2$; n_{11} as the number of 1 bits that match between $\mathbf{V}_1$ and $\mathbf{V}_2$; n_{01} as the number of 0 bits in $\mathbf{V}_1$ that match 1 bits in $\mathbf{V}_2$, and n_{10} as the number of 1 bits in $\mathbf{V}_1$ that match 0 bits in $\mathbf{V}_2$, then distance D function (or, more appropriately, match or correlation) can be computed between $\mathbf{V}_1$ and $\mathbf{V}_2$ as a function $D(n_{11}, n_{00}, n_{01}, n_{10})$. For this work, a useful class of functions (parameterized by integer S) were found to be of the form: $D(n_{11}, n_{00}, S) = (n_{11} + n_{00}/S)/512$, where $1 <= S <= 5$.

B. Weighted *k*-nn Algorithm. The weighted k-nn algorithm uses the function D as a weight in the *NN* decision procedure. The distance D is computed from $\mathbf{V}_{\text{unknown}}$ to $\mathbf{V}_k$ for $k = 1 \ldots N$ (all of the training prototypes). Next, the top k prototypes are chosen that are closest to $\mathbf{V}_{\text{unknown}}$, denoted $D(\mathbf{V}_j)$, $j = 1 \ldots k$. A weighted vote for each class C is computed as:

$$\text{Weight}_c = \frac{1}{k} \sum_{j=1}^{k} (D(\mathbf{V}_j) * \text{Class}(\mathbf{V}_j, C))$$

where Class($\mathbf{V}, C$) is an indicator function defined as:

$$\text{Class}(\mathbf{V}, C) = 0 ; \quad \text{if class label of prototype } \mathbf{V} \neq \text{class label } C$$
$$= 1 ; \quad \text{otherwise}.$$

The classes C are then ranked according to the value of Weight_C.

V. REFINEMENTS

A number of refinements of both feature generation and classification have been tried with various results. A successful improvement to the feature generation algorithm was to use a variable 4×4 sampling grid on the image. A horizontal and vertical mass histogram of the image is computed and sampling lines are placed on the equimass divisions of the histogram. This is done by dividing the total mass (M) of the image (number of black pixels) by the number of grid divisions in the x or y direction (in this case, both are 4). This average mass is denoted M_A ($=M/4$). Next, the histograms are scanned and the grid points are placed in the x and y directions such that the total mass between two adjacent points in the x or y direction is M_A. This results in higher sampling of regions with the most mass. The original sampling method can be considered as uniform distance sampling and the equimass method as uniform mass sampling. A significant improvement in performance with digit recognition has been obtained with this scheme (Fig. 5).

VI. APPLICATIONS

The GSC classifier has been used successfully in several different document reading applications: digit string segmentation and general handwritten word recognition. Each application is trained on the appropriate class data base (digits or characters) and the GSC algorithm parameters are adjusted for optimal performance.

A. Digit String Segmentation. This application consists of reading of ZIP codes from digitized postal address images. Typically, a handwritten ZIP code contains five or nine isolated digits which usually can be read by a digit recognizer. In some cases, however, two or more of the adjacent digits can touch in unpredictable says (Fig. 6). One approach for accurate recognition is to force a segmentation of the digit string and recognize each hypothesized isolated digit individually. Given the image of a string of digits, the goal of the segmentation algorithm is to partition the image into regions, each containing an isolated digit. A GSC recognition-aided iterative method is used. Adjacent digits can be touching and some of the digits might be broken into more than one component (e.g. five-hats). Therefore, the number of digits in a digit string is not simply a count of the number of components in the field. Components which are classified as touching digits must be segmented appropriately. The module which performs the segmentation and subsequent recognition of the segmented digits does the following: Given a connected component with two to four digits, the module estimates the number of digits in the component, performs the appropriate segmentations and recognizes the individual digits [11].

B. Handwritten Word Recognition. Another successful application of the GSC algorithm is in word recognition. This research develops a recognition algorithm for off-line general handwritten word recognition, where a word can be any mixture of discrete characters, cursive components, or touching discrete characters. A recognition technique called the Hypothesis Generation and Reduction algorithm (HGR) [12], and later refined as the Character Model Word Recognition (CMWR) algorithm, extracts a general nonparametric model from an unclassified word image and then finds a lexicon word which best matches this model. The basic

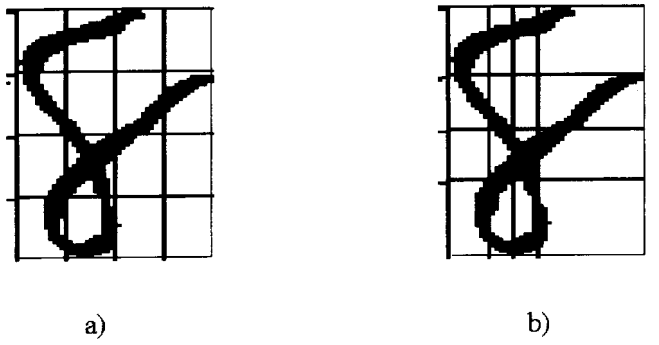

Figure 5. Sampling methods: (a) uniform distance; (b) uniform mass.

Figure 6. Digit string with touching digits.

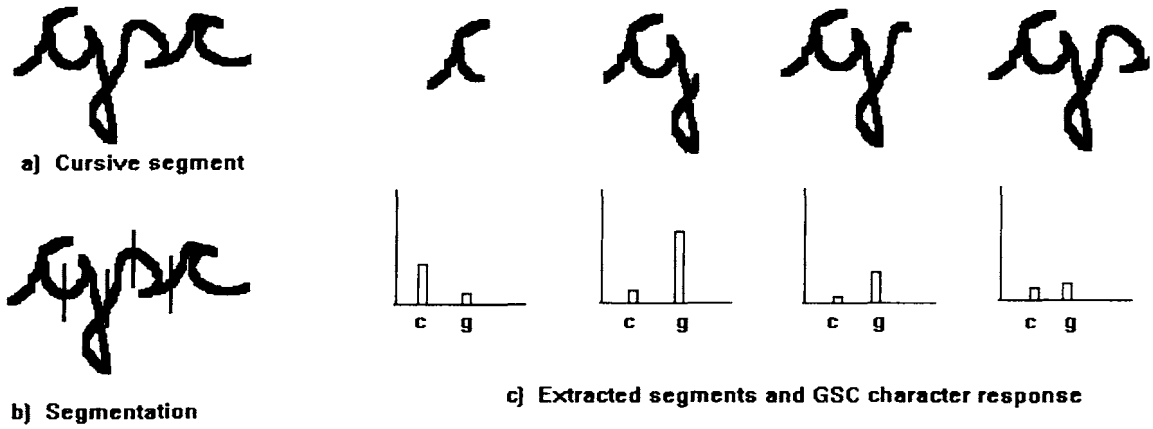

Figure 7. (a) Cursive handwriting sample; (b) segmentation points; (c) sequental extraction of segments and typical GSC response.

paradigm used to build the model is segment-and-recognize. A character oversegmentation approach is used which attempts to maximize the likelihood of generating a covering set of segmentations isolating each character of the word. This covering set is produced from the reduced output of several different segmentation algorithms, each of which uses a subset of the global image features. After segmentation, a windowed left-to-right scan of the strokes between pairs of segmentation points produces possible character interpretations of the strokes (Fig. 7). This is accomplished by carefully extracting the strokes between the points and giving it to a GSC classifier trained on mixed printed and cursive characters. The results of the GSC along with certain stroke contextual information is used to make a final character interpretation determination. A complete scan of the word in this manner produces a number of possible word hypotheses (interpretations) which can be represented compactly by a directed graph. A user-supplied lexicon and a beam search-matching algorithm constrain these hypotheses to valid words in the language. Each lexicon word is assigned a confidence which represents an average weighted distance in the underlying character feature space.

Experiments with the GSC classifier have shown that it excels in these applications because the confidences are reliable enough to use in detecting the best character and digit candidates after segmentation. Figure 8 shows the confidence distribution of the GSC algorithm tested on a series of images containing valid digits and invalid digits (fragmented digits and characters). It can be seen that the GSC algorithm offers reliable confidences in that most of the valid digits are mapped to the upper ranges of the confidence scale, while most of the invalid digits are mapped to the lower ranges.

VII. EXPERIMENTAL RESULTS

The GSC classifier has been well characterized by extensive testing. The images in these experiments were downsampled from their original resolution to 212 pixels/inch (ppi) and the training

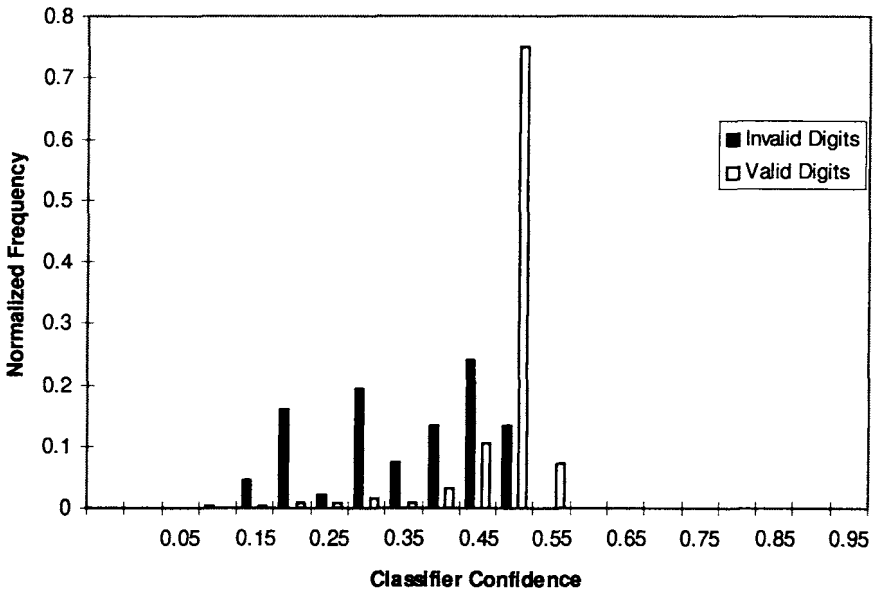

Figure 8. Confidence histograms for invalid/valid digits testset (16,000 samples).

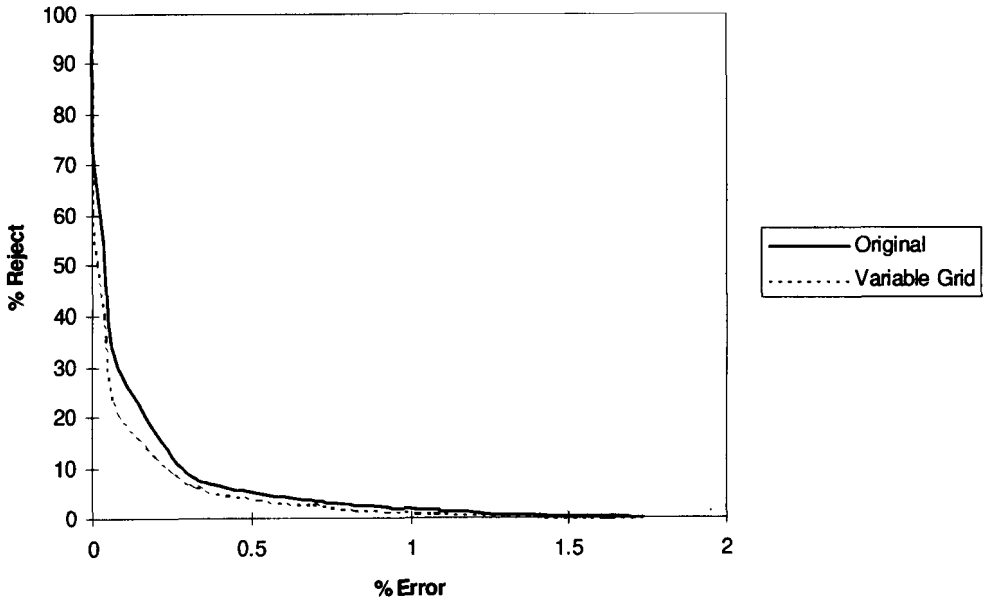

Figure 9. GSC performance on isolated digits.

sets were different from the test sets. The training/testing sets consisted of isolated digits or characters which ranged from good to poor quality. Figure 9 compares two variations of this classifier on isolated handwritten digits. The training set and testing sets were collected from two sources: handwritten postal addresses and the NIST digit data base. Figure 10 shows the GSC classifier performance trained/tested on upper-case hand-printed images. The sets used for this experiment were sampled from the NIST upper-case printed data base. Figure 11 shows the results on lower-case discrete characters sampled from the NIST lower-case printed data base. Figure 12 shows the results of the GSC classifier trained on a mix of upper/lower-case discrete and segmented cursive characters. The data sets were sampled from NIST and a data base of segmented cursive characters which were extracted from postal images. The results of these experiments are summarized in Table II.

VIII. SUMMARY

The GSC classification algorithm represents one implementation of a general philosophy of using heterogeneous features at different scales for object recognition. The scales that must be considered are local, intermediate, and large. These scales measure the important properties of the object, and the image feature detectors must be designed accordingly. The GSC algorithm detects local edge curvature, intermediate strokes, and large-scale concavities of the image object.

The separability of the objects in feature space determines the form of the classification function that can be used. Additional constraints such as the type of feature vector generated (binary, integer, and real-valued) must be considered in the practical design of an algorithm. The current GSC algorithm generates a very compact binary feature vector which allows a k-nn classifier to take advantage of bit operations of the machine architecture. One very

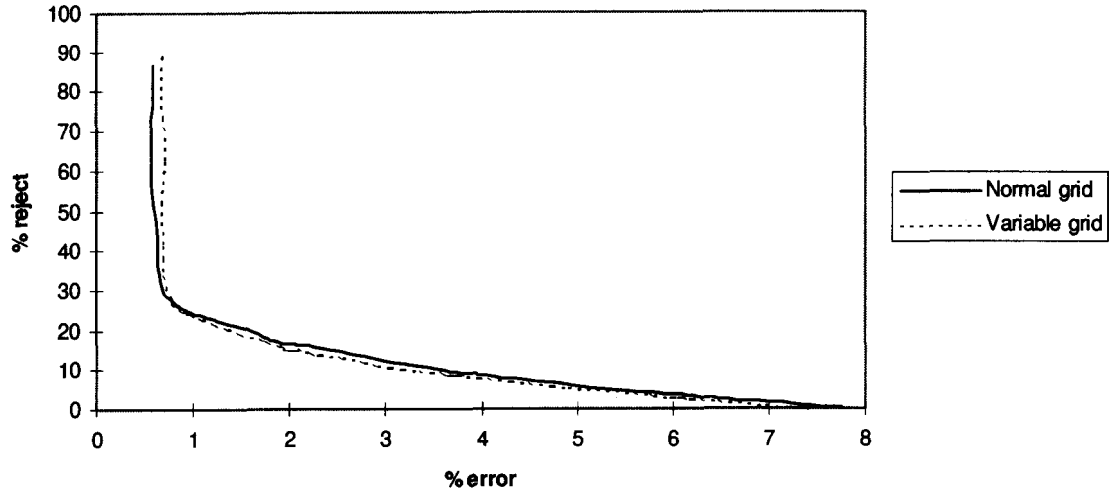

Figure 10. GSC upper-case NIST-printed character results.

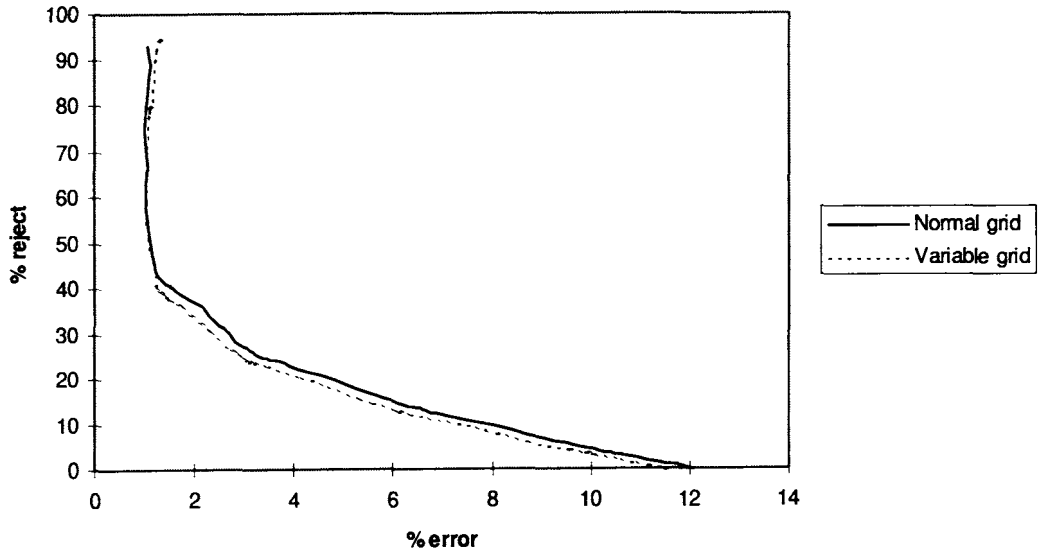

Figure 11. GSC lower-case NIST-printed character results.

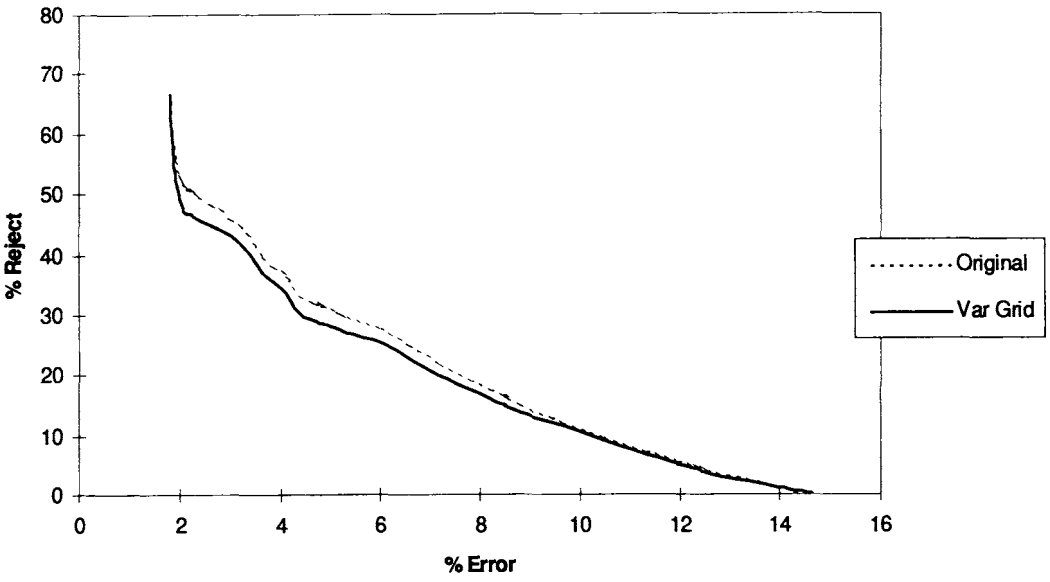

Figure 12. 26 class isolated mixed printed/cursive character results.

Table II. Summary of experiments.

			% Correct (Top Choice)	
Problem	Training Size	Testing Size	Fixed Sample Grid	Variable Sample Grid
10 class digits	24,000	12,000	98.24%	98.47%
26 class upper-case printed characters	31,000	20,000	92.15%	92.50%
26 class lower-case printed characters	34,000	23,000	87.79%	88.35%
26 class mixed-case cursive/discrete characters	40,000	7000	85.28%	86.62%

important behavior to consider is the response of the algorithm when given invalid objects which are not members of the training classes. The classifier function must have a sufficiently small response to invalid objects to allow a reject class. This property is important for applications which search an image for objects.

Experiments of this implementation on hand-printed digits, characters, and invalid objects have demonstrated its capability to recognize valid class members and reject invalid objects. This behavior has allowed it to be used successfully in document reading algorithms which look for digits or characters embedded in a set of objects.

ACKNOWLEDGMENT

The authors thank Dr. Dar-Shyang Lee for early contributions to the GSC classifier.

REFERENCES

1. C. Y. Suen et al., "Computer recognition of unconstrained handwritten numerals," *IEEE Proc.* **80**, 1162–1180 (1992).
2. G. Srikantan, "Gradient representation for handwritten character recognition," in *Proceedings of Third International Workshop on Frontiers in Handwriting Recognition (IWFHR III)*, Buffalo, NY, 1993.
3. J. Mantas, "An overview of character recognition methodologies," *Pattern Recog.* **19**, 425–430 (1986).
4. T. K. Ho, J. Hull, and S. Srihari, "Decision combination in multiple classifier systems," *IEEE PAMI* **16**, 66–75 (1994).
5. E. Kleinberg and T. K. Ho, "Pattern recognition by stochastic modeling," in *Proceedings of Third International Workshop on Frontiers in Handwriting Recognition (IWFHR III)*, Buffalo, NY, 1993.
6. D. Lee and S. Srihari, "Handprinted digit recognition: A comparison of algorithms," in *Proceedings of Third International Workshop on Frontiers in Handwriting Recognition (IWFHR III)*, Buffalo, NY, 1993.
7. J. Favata, G. Srikantan, and S. N. Srihari, "Handprinted character/digit recognition using a multiple feature/resolution philosophy," in *Proceedings of Fourth International Workshop on Frontiers in Handwriting Recognition (IWFHR IV)*, Taipei, ROC, 1994.
8. N. Otsu, "A threshold selection method from grey-level histograms," *IEEE Trans. SMC* **9**, 62–66 (1979).
9. R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis* (Wiley, New York), 1973.
10. J. D. Tubbs, "A note on binary template matching," *Pattern Recog.* **22**, 359–365 (1989).
11. R. Fenrich, "Segmentation of automatically located handwritten words," in *Proceedings of an International Workshop on Handwriting Recognition*, Bonas, France, 1991, pp. 33–44.
12. J. Favata, "Recognition of cursive, discrete, and mixed handwritten words using character, lexical and spatial constraints," Tech report 93–32, Dept. of Computer Science, SUNY Buffalo, 1993.