Multiple/Parallel Handprinted Digit Recognition

J.R. Parker
Laboratory for Computer Vision
Department of Computer Science
University of Calgary
2500 University Drive NW
Calgary, Alberta, CANADA
T2N-1N4

parker@cpsc.ucalgary.ca
(403) 220-6784
(403)284-4707 (FAX)

The use of multiple classification algorithms applied to the same input can give higher accuracy classifications than using a single algorithm. This paper discusses five schemes, two of them recently devised, for classifying handprinted digits, and shows how to merge the results from the individual classifiers into a single coherent classification.

Multiple/Parallel Handprinted Digit Recognition

The use of multiple classification algorithms applied to the same input can give higher accuracy classifications than using a single algorithm. This paper discusses five schemes, two of them recently devised, for classifying handprinted digits, and shows how to merge the results from the individual classifiers into a single coherent classification.

Summary

1. This paper contains five algorithms for recognizing handprinted characters, three of which were devised specifically for this system; a fourth method is a modification of previous work on the use of profiles [Kimura, et al, 1991]. The most important constribution is the method for merging the results from the multiple systems into a single classification, which is different from traditional approaches and works very well. Overall recognition rate was 99%+ for the test data used (1000 digits).

2. The use of multiple systems for classification is receiving more attention, and so the ability to convert various kinds of results into a common type, and to 'vote' on the most likely classification, is crucial. The example of handprinted characters (digits) is obviously of some interest, too.

3. Work by Xu/Suen,1992 and by Kimura,1991 are the most similar, since they, too, deal with multiple methods. The voting methods and the recognition methods are, however, different.

4. The use of voting methods can be applied to any system having multiple methods to accomplish the same task, and increases confidence in the result. The specific character recognition schemes have already been applied to classification of music symbols and chess pieces, and should have a general utility. Software will be made available for FTP.

5. This paper has not been published elsewhere, although parts will see print. A specific discussion of the voting methods ONLY was submitted to ANZIIS'95, and a variation on the vector template material was a poster at ICPR in Jerusalem.

There are two ways to use features to classify objects. In *statistical* approaches many features are combined into a large feature vector. Because features are measurements, the same object can correspond to a wide variety of feature vectors just through the errors in the measurements. However, these measurements will be clustered in some region of N-space, where N is the dimension of the feature vector. Hence, a statistical recognizer will construct a feature vector from a data object and classify it based on how far (Euclidean distance) it is in N-space from the feature vectors for known objects.

The basic idea behind *structural* pattern recognition is that objects are constructed from smaller components using a set of rules. Characterizing an object in an image is a matter of locating the components, which at the lowest level are features, and constructing some representation for storing the relationships between them. This representation is then checked against known patterns to see if a match can be identified. Structural pattern recognition is, in fact, a sophisticated variation on template matching, one which must match relationships between objects as well as the objects themselves.The problems involved in structural pattern recognition are two: locating the components, and finding a good representation for storing the relationships between the components.

Both methods have their strengths and weaknesses, and within each class of recognizer there are many variations of the basic theme. It is hoped that by combining many classifiers, some statistical and some structural, the resulting system will be more robust in the sense of yielding a correct classification in a wide variety of circumstances. To this end, we will examine briefly five methods for recognizing handprinted digits, and will look at methods for combining the results of all five into a single classification.

1 Properties of the Character Outline

In a couple of interesting articles, Shridhar et al [SHRI86, SHRI87, KIMU91] describe a collection of topological features that can be used to classify hand printed numerals. Most of these features are properties of the outline, or *profile*, of the numeral. For instance, a digit ‘8’ might be described as having a smooth profile on both the left and the right sides, and as having the width a minimum in the center region. Not all hand printed ‘8’ digits would be recognized by this description, and certainly some other digits might also have this description; the idea is to provide a sufficient number of descriptions for each digit that a high recognition rate can be achieved.

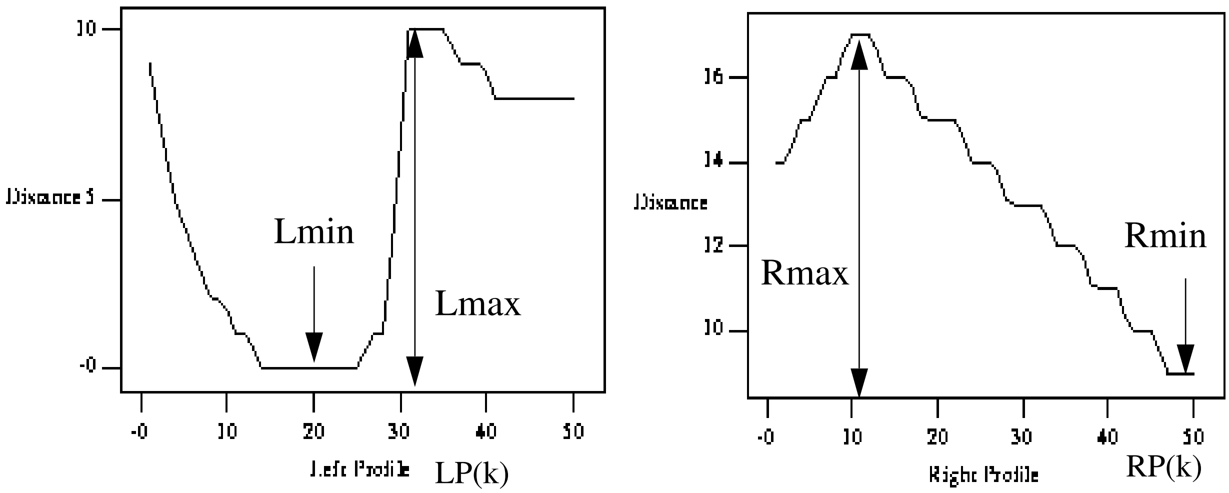


Figure 1 - Simple properties of the Left and Right profiles.

Figure 1 shows how the left and right profiles are defined and calculated for a sample digit ‘9’. After the digit is isolated and thresholded, the number of background pixels between the left side of the character’s bounding box and the first black pixel is counted and saved for each row in the bounding box. This gives a sampled version of the left profile (LP), which is then scaled to a standard size, in this case 52 pixels. A similar process gives the right profile (RP); the difference is that the last black pixel on each row is saved.

Having the profiles, the next step is to measure some of their properties. For example, one important property is the location of the extrema: L_{min} is the location of the minimum value on the left profile, and L_{max} is the location of the maximum value. of the actual peaks in LDIF and RDIF and their values happens to be quite important in characterizing numerals, and the peaks are located using a range rather than a single position. Thus, a digit ‘5’ would have the RDIF peak near the top of the digit, and the peak would have a relatively large value. This set of features is not comprehensive. In all, 48 features are used and others could be defined. (See [SHRI86] for a complete list and description)

In the training phase all 48 features are computed for each sample numeral and a feature vector is created in each case. The features are predicates (TRUE or FALSE) for example, feature number 43 is TRUE if the width of they character at row 20 is greater than or equal to the width at row 40 (W(20) >= W(40)). Then all of the resulting bit strings for each digit are searched for common elements, and the features in common for each digit class are stored in a library. Matching is performed by extracting the profiles of the input image and measuring and saving the bit string (feature vector) that results. This string is matched against the common elements of the templates - this is obviously very fast, since only bit operations are involved. A perfect match of a library bit string against an input string results in the corresponding digit class being assigned to the input digit. An example of such a bit string used for this purpose is:

1******0***********************0******0*1**1***16

This means that a ‘6’ digit has features 8,32, and 39 as FALSE and features 41, 44, and 48 as TRUE. A “*” represents a “don’t care” situation. The results from this method are only acceptable at this stage. The recognition rates for our samples are:

	0	1	2	3	4	5	6	7	8	9
% Correct	94	95	96	100	95	100	84	94	90	94
% Error	1	4	1	0	5	0	10	5	4	6
% Reject	5	1	3	0	0	0	6	1	6	0

2 Convex Deficiencies

A digitized character image consists of pixels, usually black on a background of white. Structural character recognition techniques attempt to collect the black, or object, pixels into sets that represent a feature in a model of an object to be recognized. Then an effort is made to determine the relationships between the features and to compare these against the relationships in the model, hoping to find a match. While most character recognition systems are concerned with the pixels belonging to the characters themselves, there are good arguments to be made for analyzing the size, shape, and position of the *background* regions surrounding the character image. Certainly the number and position of the *holes* has been used - an ‘8’, for example, has two holes, one in the upper part and one in the lower part of the character image, and a ‘0’ has one in the middle. However, there are other such features that might be used to classify images - for example, a numeral ‘2’ has a left-facing concave region in the top half of the image and a right-facing one in the bottom half. A more complete analysis of these *convex deficiencies* may permit the development of a classification scheme based on the background regions [PARK94d].

For use with digital character images a relatively crude but effective scheme has been developed. From each background pixel in an input image an attempt is made to draw a line in each of the four ma-

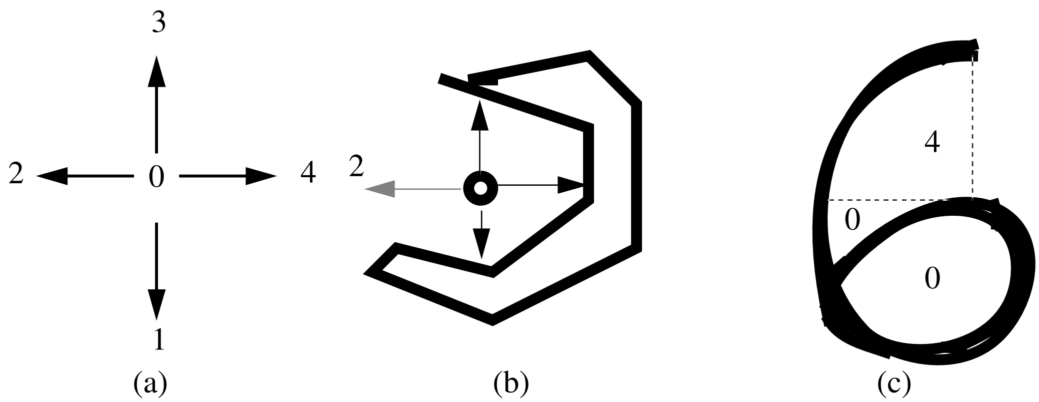


Figure 2 - Locating convex deficiencies. (a) Codes for the directions. (b) Background pixels are tested to find the open directions (this one is 2). (c) Regions found for a '6' - one of the holes (connected to the 4 region) is not real, and will be connected to another region.

jor directions (up, down, left, right). If at least three of these lines encounter an object pixel, then the original pixel is labelled with the direction in which an object pixel was *not* encountered - this is called the *open* direction. If none of the four directions are open then the pixel concerned is part of a hole, and is labelled with a zero. Figure 2 shows the direction labels, and illustrates the process of locating and labelling the convex deficiencies.

After all of the regions are labelled, they are counted and measured. Very small regions are ignored and the largest four regions are used to classify the image. Sometimes a relatively simple relationship exists; for example, 99% of all zeros can be identified by the large central hole and lack of other convex deficiencies. The next more complex scheme uses relative positions of the regions; for example, an '8' has two holes, one above the other, a left-facing region on the left of the two holes, and a right facing region on the right of the holes. The most complex schemes require shape information in addition to size and position. As an example, some '7' digits and some '3' digits both have a large left-facing region on the left side of the image. However, this region is convex for the '7' but non-convex for the '3'.

An important aspect of the method involves not being too specific about the sizes of the regions. The algorithm discards as unimportant any region that is has an area less than 10% of the object. Then regions are classified only as large or small. Position, too, is intentionally classified in only a crude manner as top, bottom, left, or right, with a special flag set for those regions that are sufficiently close to the center. This provides a description of the background geometry that is good enough to classify the image into digits, but is not overly affected by the usual variations in line thickness, orientation, size, and shape found in handprinted characters.

The recognition rate achieved using this method is acceptable, averaging about 94%. In detail:

0	1	2	3	4	5	6	7	8	9
99%	94%	98%	96%	94%	90%	90%	93%	95%	92%

3 Vector Templates

The basic idea behind template matching is that each digit has a particular shape that can be captured in a small set of models, usually stored as raster images. An incoming (unknown) digit, also in raster form, is compared against each template, and the one that matches most closely is selected as belonging to the same digit class as the unknown. Template based recognition methods work quite well for machine printed characters, which are uniform in size, shape, and orientation. On the other hand, characters printed by a human show a large degree of variation even in sets of characters printed by the same person. This variation mitigates against the use of templates.

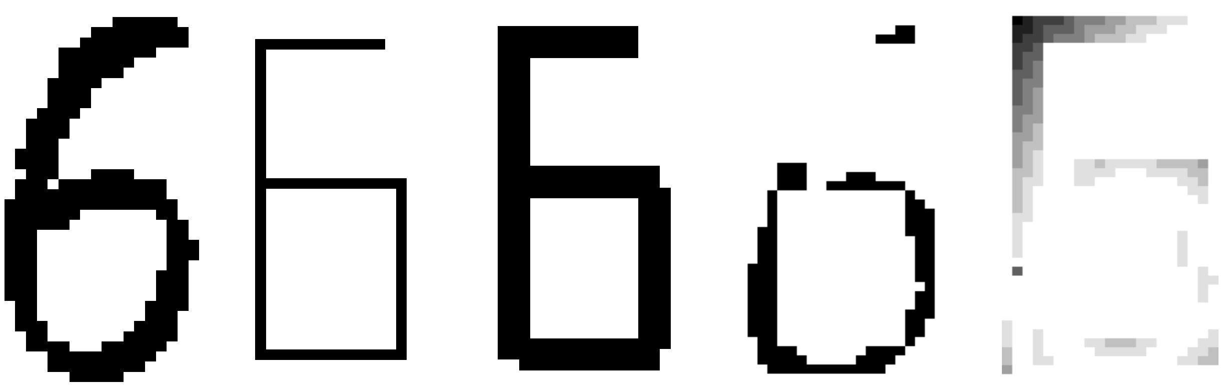
One solution is to represent the template digits as vectors[PARK94b]. The vectors that form the skeleton of the characters are used rather than the outline; this gives a good abstraction of the shape, and permits the lines to be thickened in an arbitrary way. The templates are stored as sets of four integers: the starting and ending row and column on a standard grid, and all templates have the same size. Given a scale and rotation, all templates in the collection would be modified in a consistent way.

A vector template can be produced using only a pencil, and perhaps some graph paper, and indeed, the first set of templates were generated in just this way. It is also possible to create a template from a data image, and may be desirable when starting to process data from a new source. The first step in this process is to threshold and then thin the input image. Then the pixels are collected into sets, each representing a curve. An end pixel is found (either a pixel connected only to 1 other) and the set of pixels connected to it area saved, taking care to trace only one curve[LAM]. Finally, a set of vectors is extracted from each curve using a recursive splitting technique, a relatively simple and common method for vectorizing small, simple images.

Once all of the templates have been generated, and there are multiple templates for each digit, the system is ready to recognize digits. An incoming image is first thresholded and the width of the lines is then estimated using horizontal and vertical scans. The bounding box is also found so that the templates can be scaled. Now the template vectors are drawn into an otherwise clear image the same size as the input image, producing an initial raster template that represents the scaled skeleton. Finally, each pixel is 'grown' equally on all sides to give a line width comparable to that found in the input image. The result is a raster template with some similar properties to those found in the input image. Figure 3 illustrates the process of generating a raster template from a vector one.

The matching process is somewhat different from that used in other template matching systems, but the goal is still to produce a measure of distance between the template and the image. The first step is to locate those pixels that are black in both images. These have a distance between them of zero, and are ignored in future processing. Next, each black pixel in the image has its nearest corresponding pixel in the template located and marked. The 8-distance between these pixels is noted, and a sum of these distances is computed. After all image pixels have been assigned corresponding pixels in the template the total distance is an initial measure of similarity. At this point a numeric value that can be used as a goodness of match metric has been found. It is normalized to a per-pixel distance and stored as a measure for the digit having the same class as the template. The class having the smallest such measure over all templates is chosen as the class of the input digit image.

The software implementing this method is written in C for both a Sun SPARC II workstation and an IBM PC. The time needed for template scaling and drawing is 0.019 seconds per template for a total



(a) **(b)** **(c)**

Figure 3 - Matching using a vector template. (a) Input digit image (to be matched). (b) Scaled vector template. (c) Thickened vector template (not a good match here). (d) Pixels in common between template and data image. (e) Distance map between data image and template. Dark pixels are far away, light ones are close.

of 0.55 seconds. Pixel pairing and matching takes and average of 0.015 seconds per template, or 0.84 seconds per input image. The average time to recognize a digit without using parallel processing is 4.4 seconds on a Sun SPARCstation 2, and about twice as long on the PC.

Recognition rates are relatively good:

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Correct	99%	94%	98%	96%	94%	92%	90%	93%	95%	92%

There are multiple templates for each digit, but not necessarily the same number.

4 Shape Tracing

The vector template approach essentially attempts to compare a set of standard line drawings of characters against the incoming data. A related method, which will be referred to as *shape tracing*, attempts to draw a character over top of the incoming data to see how well the drawing matches the data. With this in mind, an effort was made to characterize the motions involved in drawing all ten digits. Each motion begins at a start point, which is a point at which the pen would first touch the paper in making a stroke. Motion is specified by giving a direction only, or a series of directions for continuing the stroke. Directions are crudely specified as being one of eight possible: right-up (0-45 degrees), up-right (45-90), up-left (90-135), left-up (135-180), left-down (180-225), down-left (225-270), down-right (270-315) or right-down (315-0). Motion is approximated by straight lines; of course, curves exist in handprinted characters, but most computer software would plot a curve as a sequence of straight lines. The lines used in this recognizer are longer than would be used to plot the curve, but the basic shape information remains after the linear approximation.

An example of how to draw a digit appears in figure 4. The digit, in this case a ‘6’, begins in the upper right of the canvas., where the pen is put to paper. Drawing the digit proceeds as follows: left-down, down-left, down-right, right-down, up-right, up-left, left-up, and left-down. Not all sixes follow exactly this path, so the tracing of the path must have a certain flexibility. For example, the light grey line represents an alternate path that could just as easily occur, and should also be allowed in a six. The line lengths are not all crucial, but if all of the lines were the same length, the result would have been a zero instead of a six. Thus some effort must be made to characterize the key lines and their relative lengths.

Thus, the templates in this particular matching scheme consist of sets of pen motions, and the recognizer must attempt to measure how well each set of motions matches what is actually seen in the input image. The matching step turns out to be hard to do in software, and seven versions of the program were tested before coming up with a strategy that worked.

The procedure followed for zeros and eights is only slightly different, due to the fact that neither digit generally has a start point. If no start point is found then one is created by slicing through the glyph vertically along its own axis and using the black pixel in the centre of the first stroke encountered. A start point found in this way is special, and a note is made of this fact for later use.

Eights presented unique difficulties in tracing. There was a larger than expected variety in the shape of the eights seen in the sample data set, and that combined with the fact that an eight has a larger number of linear features than any other digit made them impossible to trace both unambiguously and reliably. The solution was, arguably, a ‘hack’, but nonetheless a reliable one. An eight will be a character

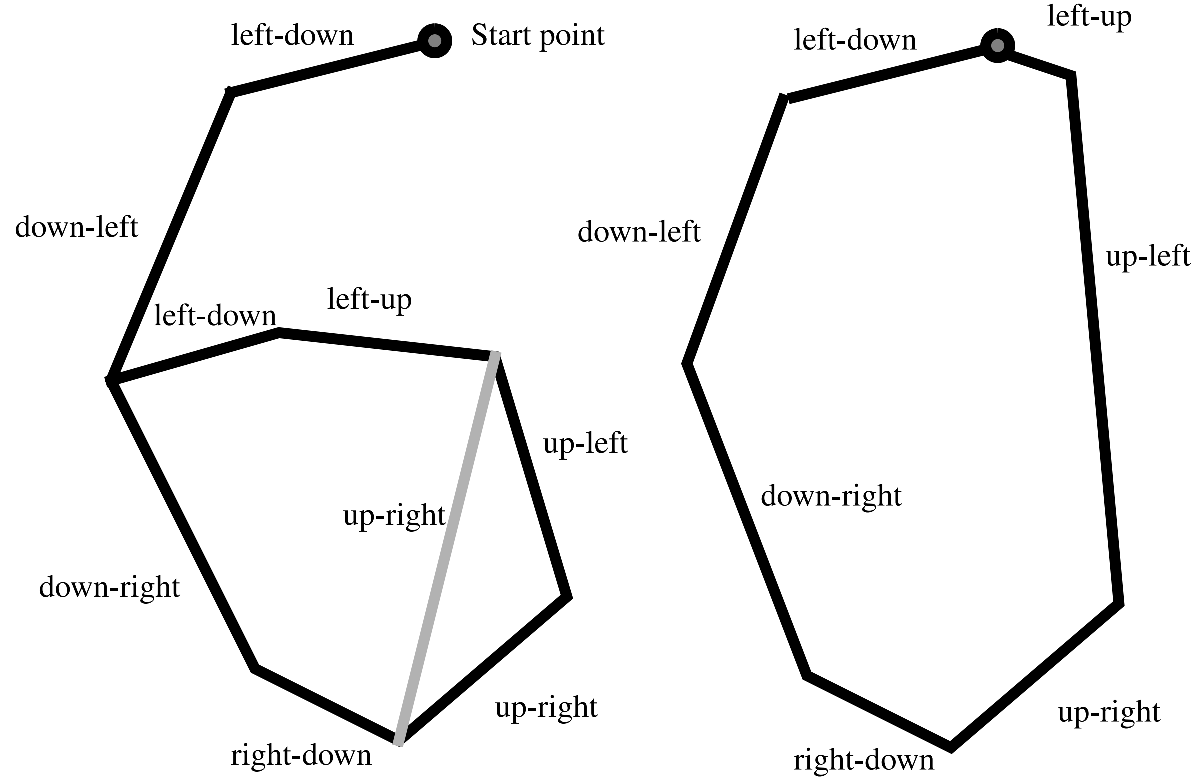


Figure 4 - Drawing a six by specifying the start point and pen motions. A zero
can be described by identical pen motions, but some of the lines are different lengths.

that more or less agrees with a zero, but has a black region in the middle. This simple process correctly classifies 95% of the ‘8’ digits.

The classifier described above was used to classify the same set of digit images used in the previous three examples and gave the following results:

	0	1	2	3	4	5	6	7	8	9
Correct	100%	94%	92%	99%	90%	94%	100%	88%	99%	98%
Reject	0	0	5%	1%	4%	3%	0	0	0	0

This gives an overall error rate of 3.3%. The low recognition rate for seven digits might be improved on in a number of ways, but is not a concern provided that it does not result in an overall low recognition rate for sevens in the multiple/parallel system.

5 Neural Nets

The use of artificial neural systems (ANS) for various recognition tasks is well founded in the literature[MUI94, SHAN94,TOUR89, YOUNG88]. The advantage of using neural nets, as they are commonly called, is not that the solution they provide is especially elegant or even fast; it is that the system ‘learns’ it own algorithm for the classification task, and does so on actual samples of data. Indeed, the same basic neural net program that recognizes digits can also be used to recognize squares, circles, and triangles. To thoroughly explore the use of neural nets is not the goal of this section nor of this thesis. Instead, the goal is to provide a fifth and last classifier to the multiple/parallel system, and one based on a different strategy than the previous methods. The hope is that the strengths and weaknesses of the methods will combine to provide a classification system having high reliability.

With this in mind, a three layer backpropagation net (BPN) is proposed. There are 48 nodes in the input layer, one node for each pixel in an image of 8 rows x 6 columns. The 3 layer net was trained using 1000 digits, 100 drawn from each class, presented alternately to the inputs; that is, one sample of each class 0 through 9 produced by a single writer was presented, followed by another set of 0 -9, and so on. If all of the 0 digits were presented first, then the 1’s and so on, the net would tend to forget the earlier digits and recognize only the later ones trained. After the training phase a second set of 1000 digits from 100 different writers was used to test the recognition rate. The results were initially not very good, at least for nines:

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
Correct	99%	93%	99%	95%	100%	95%	99%	100%	95%	74%

The logistic function gives an almost binary output that can be used as a ranking, so that a neural net of this type can provide both a simple classification and a ranking of likely classifications which can be ordered according to likelihood.

6 Converting between response types

Before proceeding to analyze methods for merging responses it would be appropriate to discuss means of converting one response type to another[XU92]. In particular, not all of the classifiers yield a rank ordering, and this will be needed before merging the responses together.

Converting a single response to a rank cannot be done in a completely general and reliable fashion. However, an approximation can be had based on the measured past performance of the particular algorithm. Each row in the confusion matrix represents the classifications actually encountered for a particular digit with that classifier expressed as a probability, and the columns represent the other classifications possible for a specified classification; this latter could be used as the confidence rating. Of course, there will be a large number of small valued and zero entries, especially if the classifier is any good, but that would be expected in any case. The conversions from type 1 can be expressed as:

Type 1 to Type 3: Compute the confusion matrix K for the classifier. If the classification in this case is j, then first compute:

Now compute the type 3 response as a vector V, where

$$V(i) = \frac{K(i, j)}{s}$$

Type 1 to Type 2: Convert from type 1 to type 3 as above, then convert to type 2 from type 3 by sorting the responses descending on likelihood.

7 Merging responses

The problem encountered when attempting to merge ranked responses is as follows: given M rankings, each having N choices, which choice has the largest degree of support? For example, consider the following 3 voter/4 choice problem [STRA80]

Voter 1: a b c d   Voter 2: c a b d   Voter 3: b d c a

This case has no majority winner; a, b and c each get one first place vote. Intuitively, it seems reasonable to use the second place votes in this case to see if the situation resolves itself. In this case b receives two second place votes to a's one, which would tend to support b as the overall choice. In the general case there are a number of techniques for merging rank-ordered votes, four of which will be discussed here.

Our analysis [PARK95] has indicated a slight advantage to the Black [BLAC58] scheme, which requires just a little explanation. The *Borda count* [BORD81] is a well-known scheme for resolving this kind of situation. Each alternative is given a number of points depending on where in the ranking it has been placed. A selection is given no points for placing last, one point for placing next to last, and so on up to N-1 points for placing first. In other words, the number of points given to a selection is the number of classes below it in the ranking. However, the Borda count does have a problem that might be considered serious. Consider the following 5 voter/3 choice problem:

Voter 1: a b c   Voter 2: a b c   Voter 3: a b c   Voter 4: b c a   Voter 5: b c a

The Borda counts are a=6, b=7, c=2, which selects b as the winner. However, a simple majority of the first place votes would have selected a! This violates the so-called *majority criterion* [STRA80]:

> ***If a majority of voters have an alternative X as their first choice, a voting rule should choose X.***

This is a weaker version of the *Condorcet Winner Criterion* [COND85]:

> ***If there is an alternative X which could obtain a majority of votes in pair-wise contests against every other alternative, a voting rule should choose X as the winner.***

With the monotonicity criterion in mind the Black[BLAC58] strategy chooses the winner by the Condorcet criterion if such a winner exists; if not, the Borda winner is chosen. This is appealing in its simplicity, and can be shown to be monotonic. The results are summarized in Table 1 rather than present the full set of confusion matrices.

Table 1: Results of the Voting Rules for Rank Ordering

Rule	Recognition	Error	Rejection	Reliability	Acceptability
Borda	99.9%	0.1	0.0	0.999	0.998
Black	99.9%	0.1	0.0	0.999	0.998
Copeland	99.6%	0.2	0.2	0.998	0.994

From this table it would appear that the Borda scheme is tied with Black's, and then Copeland's. It is important to temper this view with the fact that this result was obtained from basically one observation. Confirmation would come from applying these schemes to a large number of sets of characters. Another consideration is that a voting scheme may err in favor of the correct classification when it should, in fact, be rejected.

If the results from the classifier are simple single-valued class specifications, then the overall classification from the multiple classifier can be obtained by a simple majority vote. Based on work we have done so far, this yields the best results out of three alternative methods examined [XU92, PARK95].

Finally, in cases where actual probabilities are produced by a classifier, the overall result can be given by the class having the largest average value of all of the classes.

In order to compare the methods a single numerical value representing how good the classification is should be used. The *reliability*, expressed as recognition_rate/(100%-rejection_rate) is not very good here, since the reliability is high when the recognition rate is only 50% and the rejection rate is 50%. We used the *acceptability* measure, expressed as recognition*reliability, to compare the methods. Using this measure to assess each of the merging methods discussed, we need to look only at the best method in each of the three groups; that is, the best multiple type 1 classifier, the best type 2, and the best type three. The best three are:

Name	Type	Acceptability
SMV	values	0.994
Black	ranks	0.998
Average	probabilities	0.994

From the table above it can be seen that the best classifier in this trial uses the Black scheme for merging rank ordered responses.

8 References

[BLAC58] Black, D., **The Theory of Committees and Elections**, Cambridge University Press, 1958.

[BORD81] Borda, Jean-Charles de., **Memoire sur les Elections au Scrutin**, *Histoire de l'Academie Royale des Sciences*, Paris, 1781.

[COND85] Condorcet, Marquis de., **Essai sur l'application de l'analyse a la probabilite des decisions rendues a la pluralite des voix**, Paris, 1785.

[KIMU91] Kimura, F. and Shridhar, M., **Handwritten Numeral Recognition Based On Multiple Algorithms**, *Pattern Recognition*, Vol. 24 No 10, 1991.

[MUI94] Mui, L., Agarwal, A., Wang, P.S.P., **An Adaptive Modular Neural Network with Application to Unconstrained Character Recognition**, *IJPR*, V. 8 no 5, 1994. Pg 1189.

[PARK94a] Parker, J.R., *Practical Computer Vision Using C*, John Wiley & Sons, N.Y., 1994.

[PARK94b] Parker, J.R., **Vector Templates and Handprinted Character Recognition**, *Proc. 12th IAPR Conference on Pattern Recognition*, Jerusalem, Israel. Oct 9-13, 1994.

[PARK94c] Parker, J.R., **Recognition of Hand Printed Digits Using Multiple/Parallel Methods**, *Third Golden West International Conference on Intelligent Systems*, Las Vegas, June 6-9/94.

[PARK94d] Parker, J.R., **The Use of Convex Deficiencies for the Recognition of Hand Printed Digits**, *SPIE Vision Geometry III*, Boston, MA. Nov. 2-3, 1994. Pp. 169-175.

[PARK95] Parker, J.R., **Voting Methods for Multiple Autonomous Agents**, University of Calgary Department of Computer Science Technical report #95/558/10

[SHAN94] Shang, C. and Brown, K., **Principal Features-Based Texture Classification with Neural Networks**, *Pattern Recognition* 27, 1994, 675-687.

[SHRI86] Shridhar, M. and Badrelin, A, **Recognition of Isolated and Simply Connected Handwritten Numerals**, *Pattern Recognition*, Vol. 19 No. 1, 1986.

[SHRI87] Shridhar, M. and Badrelin, A, **Context-directed Segmentation Algorithm for Handwritten Numeral Strings**, *Image and Vision Computing*, Vol. 5 No. 1, Feb, 1

[STRA80] Straffin, P.D. Jr., **Topics in the Theory of Voting**, Birkhauser, Boston, 1980.

[TOUR89] Touretzky, D. ed., **Advances in Neural Information Processing Systems**, Morgan Kaufmann, San Mateo, CA, 1989.

[XU92] Xu, L., Krzyzak, A., and Suen, C.Y., **Methods Of Combining Multiple Classifiers And Their Application To Handwriting Recognition**, *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 22 No. 3, May/June 1992.

[YONG88] Yong, Y., **Handprinted Chinese Character Recognition via Neural Networks**, *PRL* 7, 1988, 19-25.