Recognition and Grouping of Handwritten Text in Diagrams and Equations

Michael Shilman, Paul Viola, and Kumar Chellapilla
Microsoft Research, Redmond, Washington (USA)
{shilman,viola,kumarc}@microsoft.com

Abstract

We present a framework for grouping and recognition of characters and symbols in free-form ink expressions. Initially each of the strokes on the page is linked in a proximity graph. A discriminative classifier is used to classify connected subgraphs as either making up one of the known symbols or perhaps as an invalid combination of strokes (e.g. including strokes from two different symbols). This classifier combines the rendered image of the strokes with stroke features such as curvature and endpoints. A small subset of very efficient features is selected, yielding an extremely fast classifier. Dynamic programming over connected subsets of the proximity graph is used to simultaneously find the optimal segmentation and recognition of all the strokes on the page. Experiments demonstrate that the system can achieve 94% segmentation/recognition accuracy on a test dataset containing symbols from 25 writers held out from the training process.

Keywords: symbol recognition, handwriting, segmentation, mathematics recognition

1 Introduction

Handwritten text recognition is a maturing technology that has spawned many software products. In these systems the user writes words in a structured fashion, either along a line or in an "input region". The recognition system can then process the entire line of text using dynamic programming to find the optimal recognition ***and*** segmentation of the strokes. When freed from the rigid "input region" requirement, users frequently generate free form handwritten notes which include handwritten text, diagrams, and annotation. These notes require significant initial processing in order to group the strokes into "lines" of text which can then be passed to the recognizer (see for example [10]). The grouping process is inherently difficult, and the best performance is achieved for simple paragraph structures in which there are a number of longer lines physically separated from drawing and annotations. While the grouping process could be integrated with the recognition process, the complexity of connected cursive recognition favors the two step process in which grouping precedes recognition. While it is very likely that an integration of segmentation and recognition would yield better results, the added performance does not justify the added complexity.

There are however a number of ink recognition problems which provide few constraints on the high-level layout of the page (Figure 1). One example is mathematical equation recognition, which incorporates many types of geometric layouts and symbols. Other examples include chemical structures, editing marks, musical notes, and so on. We believe that these scenarios are particularly important to pen computing because they exploit the flexibility of a pen to quickly express spatial arrangements, which is something that is currently difficult using a mouse and keyboard alone.

Therefore, we pose the problem of a symbol detector that performs integrated segmentation and classification of symbols over a page of handwritten ink. The detector should not constrain writing order, because it is common to ink extra strokes to correct characters after the fact. It should not make strict assumptions about the layout of the page. It should also scale to large pages of ink such as freeform notes, which can contain thousands of strokes in some cases.

Layout and timing-insensitive character recognition and segmentation is not an easy problem. The recognition of symbols is a well-known problem, for which many methods have been proposed [3]. The handwriting recognition community has developed countless techniques for optimizing segmentation and recognition over a fixed spatial or temporal order, and for recognizing isolated characters [8, 12]. The closest related systems are those that deal with the processing of mathematical expressions [2, 4, 6, 7, 11]. Unlike some of these systems, we are trying to solve a problem that does not require time ordering of strokes, does not require a linear organization of strokes on the page, and deals in a principled fashion with symbols that contain multiple strokes, some of which can be interpreted in isolation as another symbol.

In this paper, we propose one approach to symbol detec-

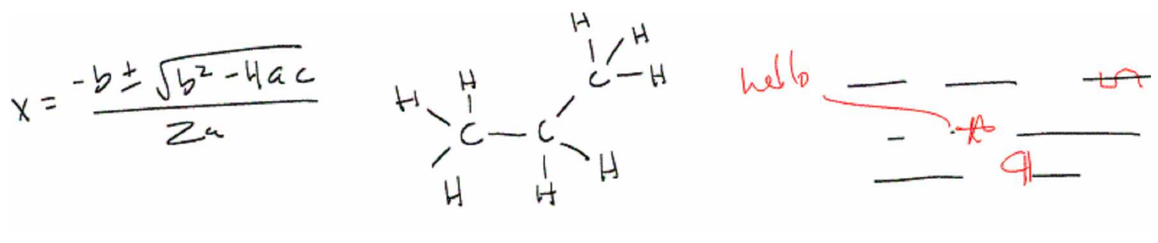

Figure 1. . The pen is a particularly useful input device when layout is unconstrained, as is the case in (a) mathematics, (b) chemistry, and (c) document annotations.

tion in online handwritten ink. The approach uses only the spatial information from the ink and the stroke segmentations. As a preprocessing step, it first builds a neighborhood graph of the ink in which nodes correspond to strokes, and edges are added when strokes are in close proximity to one another. Given this graph, we iterate efficiently over connected sets of nodes in the graph using dynamic programming and fast hashing on collections of nodes. For each set of node of up to size K, we perform a discriminative classification on the set. This allows us to incorporate non-local information that rules out spurious answers that might result from a generative model. We use dynamic programming to optimize over the space of possible explanations.

The paper is organized as follows. In Section 2 we introduce our problem as the optimization of a cost function over different groupings and recognition alternatives. We describe our optimization procedure, which leverages spatial constraints, dynamic programming, and fast classification of subsets of strokes. In Section 3 we describe our fast classifier and its features. We use a multi-class AdaBoost classifier and AMAP-style features. In Section 4, we describe our evaluation. We ran experiments over a corpus of mathematical equations and symbols whose results validate our approach. We then conclude and suggest future work.

2 Optimized Recognition and Grouping

We approach symbol recognition and grouping as an optimization problem. From a page of ink we construct a neighborhood graph $G = (V,\ E)$ in which the vertices Vcorrespond to strokes, and edges E correspond to neighbor relationships between strokes, as shown in Figure 2. From this point forward, we will use the terms *strokes* and *vertices* interchangeably.

In our system, vertices are neighbors if the minimum distance between the convex hulls of their strokes is less than a threshold. However, we expect that any reasonable proximity measure would generate similar recognition results

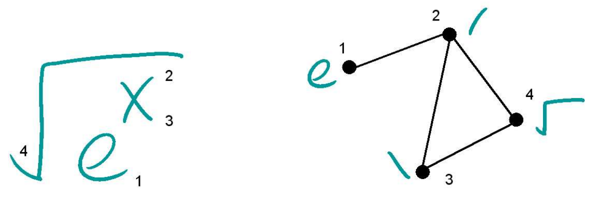

Figure 2. (a) A simple mathematical expression, and (b) its corresponding neighborhood graph.

as long as the neighborhood graph contains edges between strokes in the same symbol. We use the neighborhood graph to spatially constrain the optimization.

Based on this representation we want to find the best grouping and labeling of strokes that minimizes a global cost function:

$$C(V,E) = \min \begin{cases} & R(V, X(V,E)) \text{ if } |V| \leq K \\ \min\limits_{\substack{V' \subseteq V \\ |V'| \leq K, \\ con(V',E)}} & \Phi(R(V', X(V',E)), C(V-V', E-E' \end{cases}$$

In this function:

- $C(V,E)$ represents the best cost of grouping a set of vertices V.
- $R(V)$ is the cost of symbol recognition on that set of strokes, such as the negative log likelihood according to a classifier. We constrain all symbols to be less than a constant K strokes.
- $\Phi(c_1, c_2)$is a combination cost of two sets of nodes, such as $c_1 + c_2$. We have explored several cost functions and describe these below.

- The constraint con(V', E) requires that the vertices V' be connected in the neighborhood graph G.
- $X(V,\ E)$ is the context of the vertices, which is explained below in Section 2.3.
- E' is the subset of edges in E that touch any of the vertices in V'.

To implement this optimization efficiently, we need a way to iterate over valid sets V' (graph iteration), an efficient and accurate symbol recognizer R(recognition cost),a cost function to combine the cost of two subgraphs (combination cost), and a way to reuse computation (dynamic programming). A prerequisite for both graph iteration and dynamic programming is fast hashing on strokes, and we describe this first. In the remainder of this section we explain the rest of our optimization. In the next section we cover the details of our recognizer.

2.1 Stroke Hashing

A useful utility for both graph iteration and dynamic programming is a method for constant-time lookup of subsets of the graph. To hash on strokes, we use an XOR-based hash function on the stroke ID's. XOR is useful because it can be used to quickly add or remove strokes from an existing hash key.

$H(V) = v_0 \oplus v_1 \oplus \cdots \oplus v_N$ where $V\ =\ \{v_0, v_1, \cdots, v_N\}$
$H(V'') = H(V) \oplus H(V')$ where V" $=\ V \cup V'$

Thus we can incrementally compute the hash function on subsets of the strokes, requiring minimal computation for each lookup.

2.2 Graph Iteration

The first implementation detail of the optimization is a technique to enumerate connected subgraphs of the neighborhood graph. We wish to enumerate all connected subsets of the nodes V' in V where $|V'| \leq \quad K$. Each subset V' becomes a symbol candidate for the recognizer.

To our knowledge, there is no efficient way to enumerate subsets of up to size K without duplicating subsets. Stroke hashing allows us to quickly remove duplicate sets, so one naïve approach is to perform a bounded depth-first search from each vertex in the graph. A more sophisticated approach is to enumerate all subsets of size 1, expand each subset by all of the edges on its horizon, eliminate duplicates, expand again, and so on, up through size K. This eliminates the propagation of duplicates through each round.

Consider the graph from Figure 2. The subsets that get generated for this graph are {1}, {2}, {3}, {4}, {1, 2}, {2, 3}, {2, 4}, {3, 4}, {1, 2, 3}, {1, 2, 4}, {2, 3, 4}, {1, 2, 3, 4}.

2.3 Recognition Cost

The second implementation detail of the optimization process is the recognition cost $R(\,)$ (see Section 3 for a detailed discussion). The simple requirements are that R should return relatively low costs for subsets of the graph V' that correspond to symbols (such as in Figure 3(a)). Similarly, R should return relatively high costs for subsets of the graph that do not correspond to symbols (such as in Figure 3Figure 3**Error! Reference source not found.(b)**).

In fact, this is not as easy as it sounds. Many of the subsets V' that are passed to the recognizer are *invalid*, either containing strokes from multiple characters or do not contain all the strokes of a multi-stroke symbol. We call such subgraphs garbage. While some of the garbage doesn't look like any symbol in the training set, some invalid examples are indistinguishable from training samples without the use of context. For example a single stroke of an X can be easily interpreted in isolation as a back-slash (Figure 3(c)).

Therefore we also pass the context $X(V',\ E)$ into the recognizer to help it spot garbage. We define the context to be the set of nodes in $V - V'$ that are connected to V' in I, and show an example in Figure 3(d).

2.4 Combination Cost

The third implementation detail of the optimization is the combination cost, $\Phi(c_1,\ c_2)$. The combination cost is a function of the costs of the two subsets of the graph. We considered several alternative costs:

- **Sum.** $\Phi(c_1,\ c_2)\ =\ c_1 + c_2 + \varepsilon$. The sum of the costs makes intuitive sense: if the costs are negative log likelihoods then the sum corresponds to a product of probabilities. The ε penalty can be used to control over/under segmentation (higher values of ε force segmentation into fewer symbols).
- **Max.** $\Phi(c_1, c_2)$ = Max(c_1, c_2). This function penalizes the worst hypothesis in the set.
- **Average.** $\Phi(c_1, c_2)$ = $(c_1 + \omega c_2)$/(1+ω). This function averages the scores across all of the symbols in the hypothesis. ω is a weight corresponding to the number of symbols in the best interpretation for $V - V'$.

2.5 Dynamic Programming

Finally, Because the function we wish to optimize cleanly partitions the graph into a combination of $R(V')$

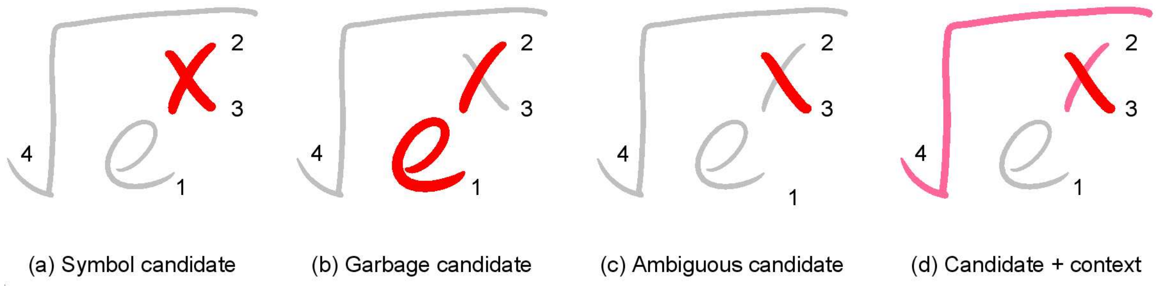

Figure 3. . Results on 3840 symbols in the context of generated mathematical expressions.

Figure 4. A set of inputs to the recognizer. (a) A full symbol that is passed as a candidate to the recognizer. (b) Undersegmented garbage that is passed as a candidate to the recognizer. (c) Garbage that is ambiguous with a back-slash when passed to the recognizer without context. (d) Neighborhood context that makes the stroke in (c) unambiguously garbage.

and $C(V - V', E - E')$, we are able to use dynamic programming to avoid redundant computation. In other words, if we have already computed C for a subset of strokes in the neighborhood graph, we can reuse the result. Using the stroke hashing technique described above, dynamic programming is simply implemented with a hash table (also known as memoization).

3 AdaBoost Discriminative Symbol Recognizer

The recognizer utilized in the dynamic programming system described above is based on a novel application of AdaBoost [5]. The primary input to the classifier is a rendered image of the strokes that comprise the hypothetical character (recall that since we do not yet know the segmentation of the strokes, strokes may not make up a character at all).

The basic framework used is most closely related to the work of Viola and Jones, who constructed a real-time face detection system using a boosted collection of simple and efficient features [13]. The Viola-Jones approach is distinguished because it classifies images extremely rapidly and reliably. We chose this approach both because of its speed and because it is easily extensible to include additional feature information.

We have generalized Viola-Jones in two ways: our classification problem is *multi-class,* and we have added addition input features to the image map. These additional features are computed directly from the on-line stroke information and include curvature, orientation, and end-point information. While we believe that this information could be computed directly from the image, this information is only currently available from on-line systems.

The input to the recognition system is a collection of images. The two principle images are the *candidate image* and the *context image.* The candidate image is quite conventional, the strokes of the current candidate sub-graph are rendered into an image which is 29x29 pixels. The geometry of the strokes is normalized so that they fit within the central 18x18 pixel region of the image. Strokes are rendered in black on white with anti-aliasing. The context image is rendered from the strokes which are connected to some candidate stroke in the proximity graph. As described above the context strokes are useful in the interpretation of multi-stoke characters such as '=' (equals), '+' (plus), and '4' (four). Since the DP process enumerates all subsets of strokes, it will generate candidates which include only 1 of the strokes in the before mentioned two stroke characters. In these cases it is the task of the recognizer to determine that this single stroke ***is not any valid symbol.*** Without context, these single strokes are easily interpretable as valid symbols (for example, the plus is comprised of a '-' (minus) and a '1' (one)). See 3.2 for an example of these principle images.

3.1 Additional Feature Images

Each of the principle images are augmented with additional stroke feature images. This is much like the earlier work on AMAP [3].1]. Each additional image is associated with a type of stroke feature which has been determined empirically. Currently there are four such images, though it is easily possible to add more. The first additional image records the curvature at each point along each stroke. The curvature is measured as the angle between the tangent at the current point and the tangent at the previous point. Both of these tangents are measured using a pair of nearby points along the stroke path that are at least 1.5 pixels distant from the current point (note that after geometrical normalization

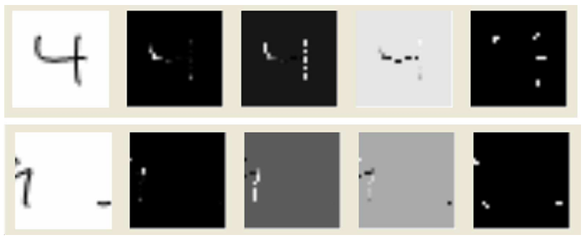

Figure 5. The left most images in each row are the candidate and context images rendered at 29x29 pixels. The remainder of each row shows various feature images computed from these principle images. From left to right: stroke curvature, stroke normal X, stroke normal Y, and endpoints.

the strokes are represented using floating point coordinates). The angle between the tangents is a signed quantity that depends on the direction of the stroke, which is undesirable. The absolute value of this angle provides direction invariant curvature information (see 3.2).

Two additional feature images measure orientation of the stroke. Orientation is a difficult issue in image processing, since it is naturally embedded on a circle (and hence 2π is identical to 0). We have chosen to represent orientation in terms of the normal vector (perpendicular vector) to the stroke (which is measured from the same nearby points used to measure curvature). The normal is a vectorial quantity and its two components are represented as two images the **normalX** image, and the **normalY** image. Note, each point on the stroke has two normals (which are opposite in direction). We choose the normal which has a positive dot product with the previous tangent (see 3.2).

The final additional feature image contains only the endpoints of the strokes, rather than the entire stroke. This measure can be useful in distinguishing two characters which have much ink in common, but have a different start and end point, or a different number of strokes (for example ‘8’ and ‘3’) (see 3.2).

3.2 The Viola-Jones Filters

The feature images described above are not used directly by the classification process, instead a very large set of simple linear function are computed from these images. These filters were proposed by Viola and Jones for the rapid detection of faces in natural images. They call each of these linear functions a ‘rectangle filter’, and each can be evaluated extremely rapidly at any scale (see 3.3). The filters measure the differences between region averages at various scales, orientations, and aspect ratios. The rigid form of these features arises from the fact that each can be computed extremely rapidly, in 6 or fewer add/multiplies. Though the form of these features makes them somewhat limited, experiments demonstrate that they provide useful information that can be boosted to perform accurate classification.

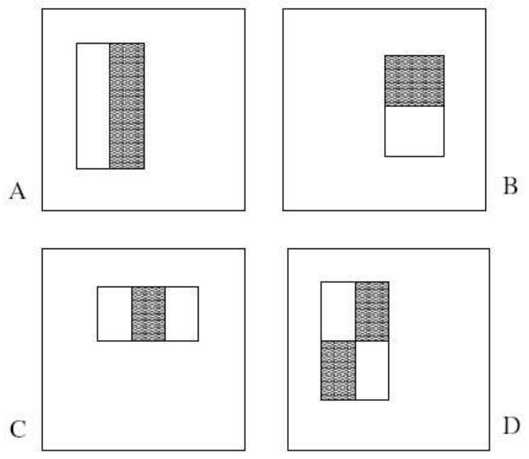

Figure 6. Example rectangle filters shown relative to the enclosing classification window. The sum of the pixels which lie within the white rectangles are subtracted from the sum of pixels in the grey rectangles. Rectangle filters which contain two rectangles are shown in (A) and (B). Figure (C) shows a three-rectangle filter, and (D) a four-rectangle filter.

For these experiments a set of one and two rectangle filters were constructed combinatorially. A set of filters of varying location, size, aspect ratio, and location are generated. The set is not exhaustive; some effort is made to minimize overlap between the filters, but it does contain 5280 filters. Such a large set is clearly overcomplete in that requires only 841 linear filters to reconstruct the original 29 by 29 image. Nevertheless this overcomplete basis is very useful for learning.

Since a rectangle filter is a function of single input image, the 5280 filters can be evaluated for each of the 10 feature images, yielding a set of 52,800 filter values for each training example. Clearly some approach for selecting a critical subset of these will improve performance.

3.3 AdaBoost Feature Selection and Learning

The above sections describe a processing pipeline for training data: a rendering process for candidate and context, a set of additional feature images, and set of rectangle filters. The machine learning problem is to generate a classifier for this data which correctly determines the correct symbol of the candidate strokes, or possibly that the set of strokes is garbage. We use AdaBoost to learn a classifier which selects a small set of rectangle filters and combines them.

For these experiments the "weak learner" is a classifier which selects a single rectangle filter and applies a threshold (this is a type of decision tree known as a decision stump). In each round of boosting the single best stump is selected, and then the examples are reweighted. We use the multi-class variant of confidence rated boosting algorithm proposed by Schapire and Singer [9].

After N rounds, the final classifier contains N weak classifiers. Since each weak classifier depends on a single rectangle filter only N filters need to be evaluated. Excellent performance is achieved with between 75 and 200 filters. On a training set of 3800 examples from 25 writers, 0 training errors is observed with 165 weak classifiers. On a test set of 3800 examples from a different set of 25 writers 96% of the characters were classified correctly.

4 Evaluation

To evaluate our approach, we ran tests on a corpus of automatically-generated mathematical expressions. We collected a very modest set of handwritten characters, digits, and mathematical operators from 50 users with 5 examples per class. Of these examples, we synthesized short expressions containing digits and operators with a generative grammar (Figure 6). Our generated expressions are intentionally dense, in order to make the segmentation problem more interesting. Also it is worth noting that although each of our test examples is horizontally-oriented, our technique applies independent of the layout. We have manually applied the technique to examples with more interesting layouts and show that it works in practice, although our test data does not reflect this condition.

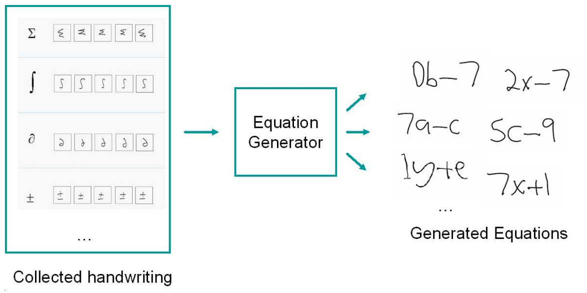

Figure 7. Collected truth data is rendered into two sets of mathematical expressions, which serve as training and test data, respectively.

We separated the generated expressions into training and test data, such that 25 users' data made up the training set and the other 25 users made up the test set. This split ensures that we are testing the generalization of the classifier across different populations.

We applied the above system to the test data with three different combination cost functions: *sum*, *max*, and *avg*, as described in Section 2.4. For *sum* we varied the value of ε to see its effect on the overall accuracy. For all of these approaches we measured the total number of symbols in the test data, the total number of false positives and false negatives in the results. A false negative occurs any time there is a group of strokes with a specific symbol label in the test data, and that exact group/label does not occur in the test data. A false positive is the converse. Our best results are 94% accuracy for segmentation and labeling for the *avg* combination cost. The full results are shown in Figure 3.

Cost	False Pos	False Neg	Total
Sum (ε = -.2)	299	243	3840
Sum (ε = -.25)	308	248	3840
Max	267	243	3840
Avg	225	202	3840

5 Conclusion

This paper presents an integrated segmentation and recognition system of on-line freeform ink. Segmentation, or equivalently grouping, is a requirement for recognition in such tasks because each symbol may have a number of strokes. Simple heuristics that group intersecting strokes may work in some cases. In domains which include multi-stroke symbols such as '=' (equals) or 'π' (pi), these heuristics fail. Conversely, it is not uncommon to see strokes from different characters come very close to or intersect each other.

This integrated system first constructs a proximity graph which links pairs of strokes if they are sufficiently close together. The system then enumerates all possible connected subgraphs looking for those that represent valid characters. The notion of proximity is defined so that strokes from the same symbol are always connected. This definition of proximity will necessarily link strokes from neighboring symbols as well. These connected subgraphs are not interpretable as a valid symbol, and will be discarded as garbage. Note, a garbage subgraph can also arise if a symbol is oversegmented: e.g. only one of the strokes in a multi-stroke character is included. A fast classifier based on AdaBoost is trained to recognize all symbol classes as well as a unique class called garbage, which includes subgraphs of strokes that are not valid. In order to address the oversegmentation problem, the classifier operates both on the current candidate strokes as well as the context of the surround strokes.

Dynamic programming is used to search for the minimum cost decomposition of the initial proximity graph into connected subgraphs, each of which can be interpreted as a valid symbol. The set of all possible connected subgraphs is efficiently enumerated using an incremental hashing scheme which grows subgraphs one node at a time and efficiently removes duplicates.

The recognizer is trained on symbols which come from 25 writers. The final system achieves a 94% simultaneous segmentation and recognition rate on test data from 25 different users which was not used during training.

6 References

1. Y. Bengio and Y. LeCun, ”Word Normalization for On-line Handwritten Word Recognition,” in Proc. of the International Conference on Pattern Recognition, (IAPR, ed.), (Jerusalem), pp. 409-413, Oct. 1994. 5 pages.

2. D. Blostein and A. Grbavec, ”Recognition of Mathematical Notation,” in Handbook of Character Recognition and Document Image Analysis, Eds. H. Bunke and P. Wang, World Scientific, 1997, pp. 557-582.

3. A. Chhabra. “Graphic Symbol Recognition: An Overview.” In Proceedings of Second International Workshop on Graphics Recognition, Nancy (France), pages 244–252, August 1997.

4. K. Chan and D. Yeung, ”Mathematical Expression Recognition: A Survey,” Int’l J. Document Analysis and Recognition, vol. 3, no. 1, pp. 3-15, Aug. 2000

5. Y. Freund, R. Schapire. “Experiments with a New Boosting Algorithm.” ICML 1996: 148-156

6. N. Matsakis, ”Recognition of Handwritten Mathematical Expressions”, Massachusetts Institute of Technology, Cambridge, MA”, May 1999

7. E.G. Miller and P.A. Viola, ”Ambiguity and Constraint in Mathematical Expression Recognition,” Proc. 15th Nat’l Conf. Artificial Intelligence, pp. 784-791, 1998

8. R. Plamondon, S. Srihari. “On-Line and Off-Line Handwriting Recognition: A Comprehensive Survey.” IEEE Transactions on Pattern Analysis and Machine Intelligence 22(1): 63-84 (2000)

9. R. Schapire, Y. Singer. “Improved Boosting Algorithms using Confidence-Rated Predictions.” COLT 1998: 80-91

10. M. Shilman, Z. Wei, S. Raghupathy, P. Simard, D. Jones: “Discerning Structure from Freeform Handwritten Notes.” ICDAR 2003: 60-65

11. S. Smithies, K. Novins, and J. Arvo, “A Handwriting-Based Equation Editor,” *Graphics Interface ‘99,* June 1999.

12. C. Tappert, C. Suen, T. Wakahara. “The State of the Art in Online Handwriting Recognition.” IEEE Transactions on Pattern Analysis and Machine Intelligence 12(8): 787-808 (1990)

13. P. Viola, M. Jones: Robust Real-Time Face Detection. ICCV 2001: 747