Comparison of Classifier Methods: A Case Study in Handwritten Digit Recognition

Léon Bottou*, Corinna Cortes, John S. Denker, Harris Drucker, Isabelle Guyon, L. D. Jackel,
Yann LeCun, Urs A. Müller†, Eduard Säckinger, Patrice Simard, and Vladimir Vapnik

AT&T Bell Laboratories, Holmdel, NJ 07733
*Neuristique, 24 rue des Petites Ecuries, 75010 Paris, France
†Electronics Laboratory, Swiss Federal Institute of Technology,
ETH Zentrum, CH-8092 Zürich, Switzerland

Abstract

This paper compares the performance of several classifier algorithms on a standard database of handwritten digits. We consider not only raw accuracy, but also training time, recognition time, and memory requirements. When available, we report measurements of the fraction of patterns that must be rejected so that the remaining patterns have misclassification rates less than a given threshold.

1: Introduction

Great strides have been achieved in pattern recognition in recent years. Particularly striking results have been attained in the area of handwritten digit recognition. This rapid progress has resulted from a combination of a number of developments including the proliferation of powerful, inexpensive computers, the invention of new algorithms that take advantage of these computers, and the availability of large databases of characters that can be used for training and testing.

At AT&T Bell Laboratories we have developed a suite of classifier algorithms. In this paper we contrast the relative merits of each of the algorithms. In addition to accuracy, we look at measures that affect implementation, such as training time, run time, and memory requirements.

2: Databases

We begin by describing the databases we have used for the benchmark measurements described in this paper.

2.1: The NIST test set

Responding to the community's need for better benchmarking, the US National Institute of Standards and Technology (NIST) provided a database of handwritten characters on 2 CD ROMs. NIST organized a competition based on this data in which the training data was known as *NIST Special Database 3*, and the test data was known as *NIST Test Data 1*.

After the competition was completed, many competitors were distressed to see that although they achieved error rates of less than 1% on validation sets drawn from the training data, their performance on the test data was much worse. NIST disclosed that the training set and the test set were representative of different distributions: the training set consisted of characters written by paid US census workers, while the test set was collected from characters written by uncooperative high school students. Examples from these training and test sets are shown in Figure 1. Notice that the test images contain some very ambiguous patterns. Although this disparity in distributions is certainly possible in a real world application, it is prudent (and usually possible) to guard against it. In general we can expect best test results when recognizers are tuned to the kind of data they are likely to encounter when deployed.

A more subtle, but, for us, a more serious problem arises from having the training and test data belonging to different distributions. Most of our machine learning techniques now use the principles of Structural Risk Minimization [1] in which the *capacity* (roughly speaking, the

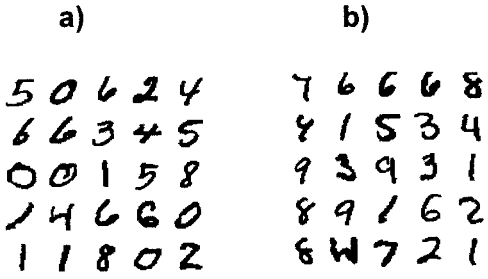

Figure 1 a) Typical images from the NIST training set, and b) Typical images from the NIST test set.

1051-4651/94 $04.00 © 1994 IEEE

number of free parameters) of a classifier is adjusted to match the quantity and the complexity of the training data. Because of the difference in distributions, we cannot use our full machine learning tool set on the NIST data when it is partitioned in this way.

2.4: Modified NIST (MNIST) training and test sets

For the reasons described above, we repartitioned the NIST data to provide large training and test sets that share the same distribution. We now describe how our new database was created.

The original NIST test contains 58,527 digit images written by 500 different writers. In contrast to the training set, where blocks of data from each writer appear in sequence, the data in the NIST test set is scrambled. Writer identities for the test set is available and we used this information to unscramble the writers. We then split this NIST test set in two: characters written by the first 250 writers went into our new training set. The remaining 250 writers were placed in our test set. Thus we had two sets with nearly 30,000 examples each. The new training set was completed with enough examples from the old NIST training set, starting at pattern # 0, to make a full set of 60,000 training patterns. Similarly, the new test set was completed with old training examples starting at pattern # 35,000 to make a full set with 60,000 test patterns.

All the images were size normalized to fit in a 20 x 20 pixel box, and were then centered to fit in a 28 x 28 image using center of gravity. Grayscale pixel values were used to reduce the effects of aliasing. These are the training and test sets used in the benchmarks described in this paper. In this paper, we will call them the MNIST data.

3: The classifiers

In this section we briefly describe the classifiers used in our study. For more complete descriptions readers may consult the references.

3.1: Baseline linear classifier

Possibly the simplest classifier that one might consider is a linear [2] classifier. Each input pixel value contributes to a weighted sum for each output unit. The output unit with the highest sum (including the contribution of a bias constant) indicates the class of the input character. In this kind of classifier there are 10 N weights + 10 biases, where N is the number of input pixels. For our 28 x 28 input units, we have 7850. Because this linear problem optimizes a quadratic function it has a single minumum with a unique solution. This means that the weight values can be determined uniquely. The deficiencies of the linear classifier are well documented [3]and it is included here simply to form a basis of comparison for more sophisticated classifiers. On the MNIST data the linear classifier achieved 8.4% error on the test set.

3.2: Baseline nearest neighbor classifier

Another simple classifier is a k-nearest neighbor classifier with a Euclidean distance measure between input image pixel maps. This classifier has the advantage that no training time is required. However, the memory requirement is large: the entire training database, about 30 Megabytes, must be available at run time. MNIST test set error for k=3 is 2.4%.

3.3: LeNet 1

LeNet 1 is a multilayer neural network that performs successive non-linear convolutions and subsampling to automatically extract relevant features [4]. Although about 140,000 multiply/add steps are required to evaluate LeNet, its convolutional nature keeps the number of free parameters to only ~3000. The LeNet 1 architecture was developed using a postal database that is smaller than MNIST database and its size was tuned to match the available data. On the MNIST LeNet 1 achieved 1.7% error.

3.4: LeNet 4

LeNet 4, which was designed for the larger MNIST database, is an expanded version of LeNet 1 that includes more feature maps and an additional layer of hidden units that is fully connected to both the last layer of features maps and to the output units. LeNet 4 requires about 260,000 multiply/add steps and has about 17,000 free parameters. LeNet 4 achieves 1.1% error on the MNIST test.

3.5: Large fully-connected multi-layer neural network

Another classifier that we tested was a fully connected multi-layer neural network with two layers of weights. Best results were obtained with 300 hidden units. For this network, the search for the optimal number of hidden units was aided by use of the MUSIC [5] supercomputer. (For purposes of comparison, numbers quoted in Figure 3 are for equivalent times on a Sparc 10.) This classifier attains 1.6% error on the test set.

3.6: Boosted LeNet 4

Several years ago, Schapire [6] proposed methods (called "boosting") for building a committee of learning machines that could provide increased accuracy compared to a single machine. Drucker, et. al. [7] expanded on this concept and developed practical algorithms for increasing the performance of a committee of three learning machines. The basic method works as follows: One machine is trained the usual way. A second machine is trained on patterns that are filtered by the first machine so that the second machine sees a mix of patterns, 50% of which the

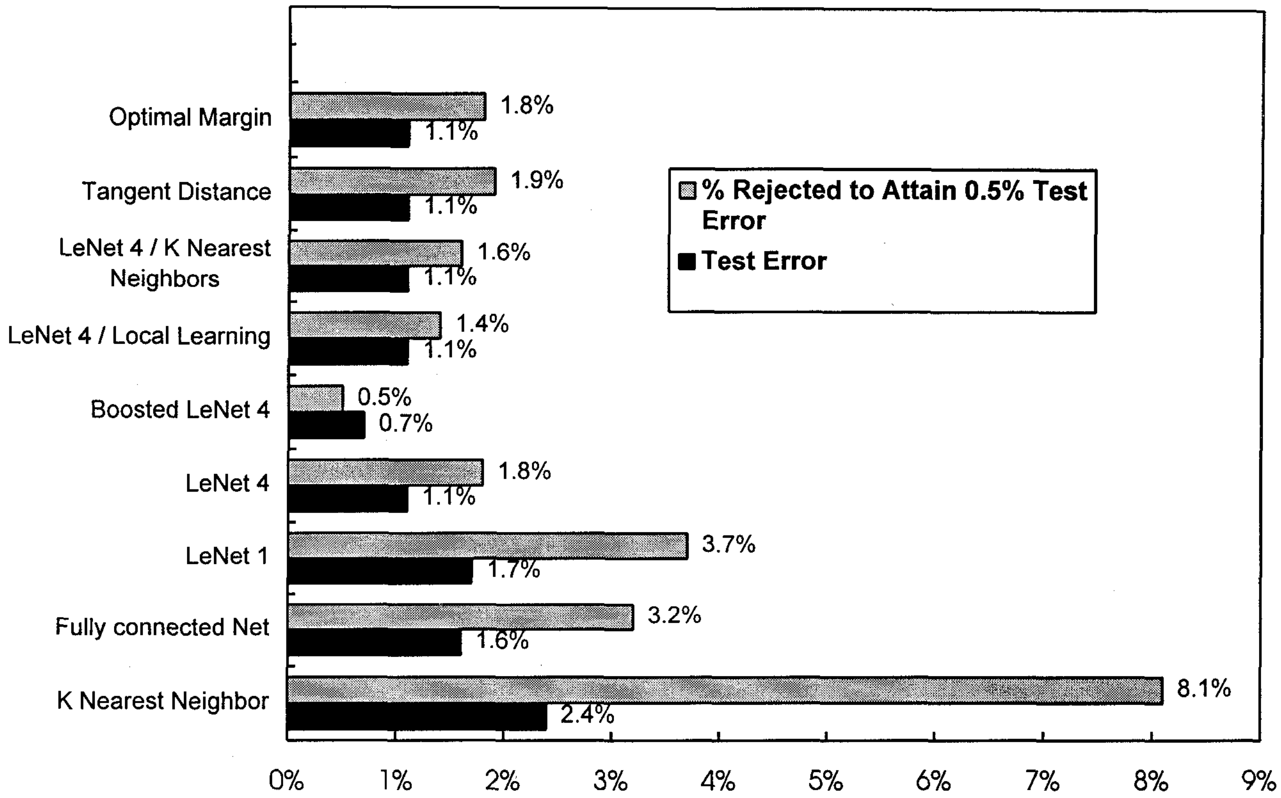

Figure 2. Performance of classifiers on the MNIST test set. The uncertainty in the quoted rates is about 0.1%. The black bars show the error rates. The error rate for the simple linear classifier (not shown) is 8.4% The gray bars show the percent of test patterns rejected to achieve 0.5% error on the remaining test examples. Results are not available for the linear classifier and the fully connected net.

first machine got right and 50% of which it got wrong.

Finally, a third machine is trained on new patterns on which the first and second machines disagree. During testing, in the Drucker method, all three machines are shown the unknown character and their output scores are added, with the highest total score indicating the most likely classification.

Notice that if the first machine is a version of LeNet 4, its ~1% error rate means that an enormous amount of data must be filtered to glean enough mis-classified patterns to train a second machine that is as complex as LeNet 4. Even more data is required to train the third machine. For this MNIST database there was insufficient data to train all three machines. In order to circumvent this problem, an unlimited number of training patterns was generated by deforming the training data with a set of affine transformations and line-thickness variations. This choice of distortions, in effect, builds some of our knowledge about character recognition into the training process. With this method, a composite machine, consisting of three versions of LeNet 4, was trained. It attained a test error rate of 0.7%, the best of any of our classifiers. At first glance, boosting appears to require three times as much time to perform recognition as a single machine. In fact, with a simple trick, the additional computation cost is only about a factor of 1.75. This is because usually the first machine classifies patterns with high confidence and the outputs of the other two machines need not be evaluated.

3.7: Tangent Distance Classifier (TDC)

The TDC is a memory-based, k-nearest-neighbor classifier in which test patterns are compared to labeled, prototype patterns in the training set. The class of the training pattern "closest" to the test pattern indicates the class of the test pattern. The key to performance is to determine what "close" means for character images. In simple nearest-neighbor classifiers, Euclidean distance is used: we simply take the squares of the difference in the values of corresponding pixels between the test image and the prototype pattern. The flaw in such an approach is apparent: a misalignment between otherwise identical images can lead to a large distance.

Simard [8] and his coworkers realized that a better distance measure should be invariant against small distortions, including line thickness variations, translations, rotations, scale change, etc. If we consider an image as a point in a high dimensional pixel space where the dimensionality equals the number of pixels, then an evolving distortion of a character traces out a curve in pixel space. Taken together, all these distortions define a low-dimensional manifold in pixel space. For small distortions, in the vicinity of the original image, this manifold can be approximated by a plane, known as the *tangent plane.* Simard, et. al. found that an excellent measure of "closeness" for character images is the distance between their tangent planes. Using this "tangent distance", a high accuracy classifier was crafted for use on the postal data. On the MNIST data a TDC with k=3 achieved 1.1% error.

3.8: LeNet 4 with K-Nearest Neighbors

As an alternative to a smart distance measure like the one used in the TDC, one can seek a change in representation so that Euclidean distance is a good measure of pattern similarity. We realized that the penultimate layer of LeNet 4, which has 50 units, can be used to create a feature vector that is appropriate for a Euclidean distance search. With these features, a 1.1% test error was attained, the same as LeNet 4.

3.9: Local Learning with LeNet 4

Bottou and Vapnik [9] employed the concept of local learning in an attempt to get higher classifier accuracy. They had observed that the LeNet family of classifiers performs poorly on rare, atypical patterns, and interpreted this behavior as a capacity control problem. They surmised that the modeling capacity of the network is too large in areas of the input space where the patterns are rare and too small in areas where patterns are plentiful. To alleviate this problem they decided to train simple linear classifiers which operate on feature vectors produced by the penultimate layer of LeNet 4. Local training uses only the *k* patterns in training set that are closest to the test pattern. In order to control the capacity of these linear classifiers, they imposed a weight decay parameter γ. The parameters *k* and γ are determined by cross validation experiments. With this local learning approach, an error rate of 1.1% was achieved on the MNIST test, the same as LeNet 4.

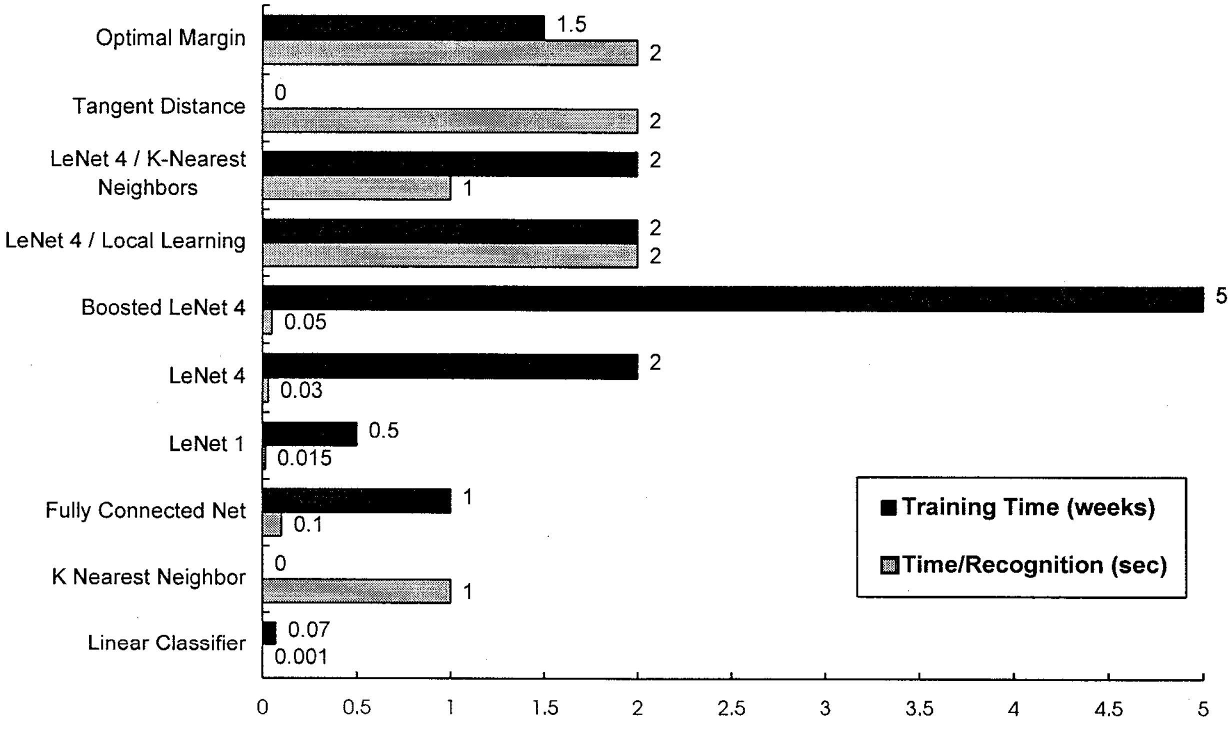

Figure 3. The black bars show the training time (in weeks) on a Sparc 10 . The gray bars show the time on a Sparc 10 for recognition of a single character starting with a size-normalized pixel map image.

3.10: Optimal Margin Classifier (OMC)

The Optimal Margin Classifier (OMC) is a method for constructing decision rules for two-group pattern classification problems. (For digit recognition, 10 such classifiers are constructed, each one checking for the presence of a particular digit.) The OMC can accommodate arbitrarily shaped decision surfaces. This is achieved by automatically transforming the input patterns and constructing a linear decision surface in the transformed space.

In the transformed space, only some of the initial patterns are required to define the decision boundaries. These are known as the *support patterns*. Only support patterns need be stored, so the memory requirements of the OMC is less than a memory-based classifier that stores all the training patterns.

The original OMC algorithm, developed by Boser, Guyon, and Vapnik**[10]**, only succeeds if the training set is linearly separable in the transformed space. The technique was extended by Cortes and Vapnik to cover inseparability, and thus allows for labeling errors in the training set **[11]**. The test results reported here make use of a 4th degree polynomial decision surface in the input space. A MNIST test error of 1.1% was obtained.

4: Discussion

A summary of the performance of our classifiers is shown in Figures 2-4. The black bars in Figure 2 show the raw error rate of the classifiers on a 10,000 example test set. Although all the classifiers, with the exception of the simple linear classifier, did well on the test set, Boosted LeNet 4 is clearly the best, achieving a score of 0.7%. This can be compared to our estimate of human performance, 0.2%. The gray bars in Figure 2 illustrate another measure of accuracy, namely the number of patterns in the test set that must be rejected to attain a 0.5% error on the remaining test examples. In many applications, rejection performance is more significant than raw error rate. Again, Boosted LeNet 4 has the best score.

Classification speed is also of prime importance. The black bars in Figure 3 show the time required on a Sparc 10 for each method to recognize a test pattern starting with a size-normalized pixel map image. Here we see that there is an enormous variation in speed. The times shown in Figure 3 represent reasonably well-optimized code running on general purpose hardware. Using special purpose hardware, much higher speeds might be attained, provided that the hardware matches the algorithm. Single-board hardware designed with LeNet 1 in mind performs recognition at 1000 characters/sec [12].

Another measure with practical significance is the time required to train the classifiers. For the local learning, training time is dominated by the time required to train a version of LeNet 4 which produces the feature vectors needed for this method. For the other algorithms, again there is significant variation in the training time. The gray bars in Figure 3 show the required training on a Sparc 10 measured in weeks.

Figure 4 shows a further measure of performance: the memory requirements of our various classifiers. Clever compression of the data or elimination of redundant training examples might reduce the size requirements of the memory-based classifiers that we tested -- at the cost of increased run time. Of the high-accuracy classifiers, LeNet 4 requires the least memory.

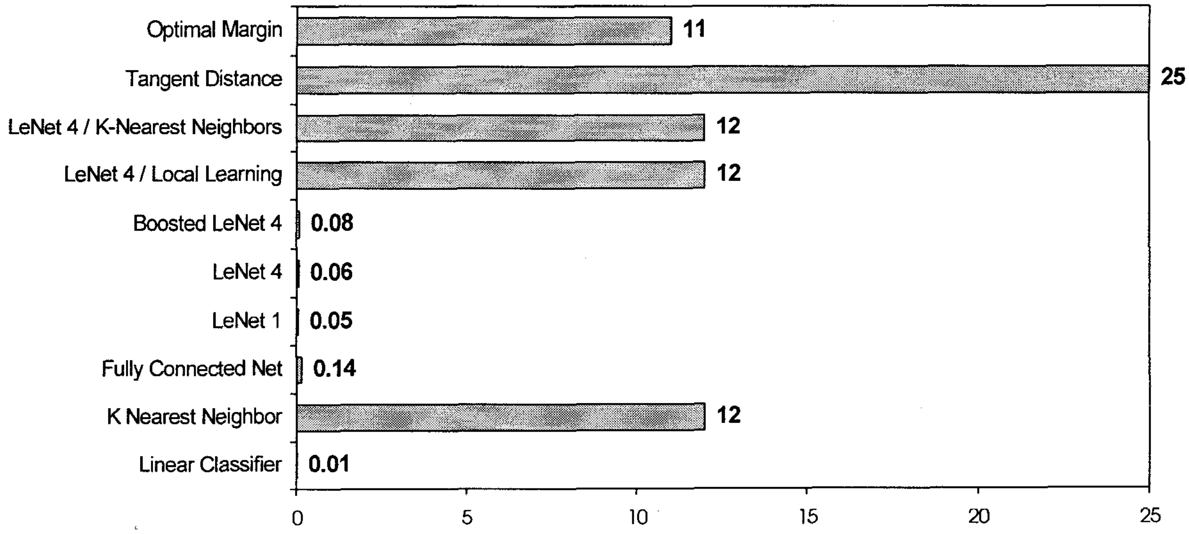

Figure 4. Memory requirements in Megabytes for different classifiers.

Many real-world applications require a multi-character recognizer. This can be implemented as a number of single-character recognizers in conjunction with an alignment lattice. The recognizers must be designed and trained to find not only the correct character (as discussed above), but also the correct segmentation [13]. We find that neural networks have a big advantage over memory-based techniques, because the latter cannot easily make use of information about counterexamples.

5: Conclusions

This paper is a snapshot of ongoing work. Although we expect continued changes in all aspects of recognition technology, there are some conclusions that are likely to remain valid for some time.

Performance depends on many factors including high accuracy, low run time, low memory requirements, and reasonable training time. As computer technology improves, larger-capacity recognizers become feasible. Larger recognizers in turn require larger training sets. LeNet was appropriate to the available technology five years ago, just as LeNet 4 is appropriate now. Five years ago a recognizer as complex as LeNet 4 would have required several months' training, and was therefore not even considered.

For quite a long time, LeNet 1 was considered the state of the art. The local learning classifier, the optimal margin classifier, and the tangent distance classifier were developed to improve upon LeNet 1 -- and they succeeded at that. However, they in turn motivated a search for improved neural network architectures. This search was guided in part by estimates of the capacity of various learning machines, derived from measurements of the training and test error (on the large MNIST database) as a function of the number of training examples. We discovered that more capacity was needed. Through a series of experiments in architecture, combined with an analysis of the characteristics of recognition errors, LeNet 4 was crafted.

We find that boosting gives a substantial improvement in accuracy, with a relatively modest penalty in memory and computing expense. Also, distortion models can be used to increase the effective size of a data set without actually taking more data.

The optimal margin classifier has excellent accuracy, which is most remarkable, because unlike the other high performance classifiers, it does not include knowledge about the geometry of the problem. In fact, this classifier would do just as well if the image pixels were encrypted, e.g., by a fixed, random permutation.

When plenty of data is available, many methods can attain respectable accuracy. Although the neural-net methods require considerable training time, trained networks usually run much faster and require much less space than memory-based techniques. The neural nets' advantage will become more striking as training databases continue to increase in size.

6: References

1. V. N. Vapnik, **Estimation of Dependencies Based on Empirical Data**, Springer-Verlag (1982).

2. R. O. Duda and P. E. Hart, **Pattern Classification and Scene Analysis**, Chapter 4, John Wiley and Sons (1973).

3. M. L. Minsky and S. Pappert, **Perceptrons**, MIT Press, Cambridge Mass. (1969).

4. Y. LeCun, O. Matan, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel, and H. S. Baird, "Handwritten Zipcode Recognition With Multilayer Networks," *Proc. of International Conference on Pattern Recognition, Atlantic City*, 1990.

5. U. A. Müller, Bernhard Baumle, P. Kohler, A. Gunzinger, and W. Guggenbuhl, "Achieving Supercomputer Performance for Neural Net Simulation with an Array of Digital Signal Processors," *IEEE Micro Magazine*, October 1992 55-65.

6. R. Schapire, "The Strength of Weak Learnability," *Machine Learning* **5** 197-227 (1990).

7. H. Drucker, R. Schapire, and P. Simard, "Boosting Performance in Neural Networks," *International Journal of Pattern Recognition and Artificial Intelligence* **7** 705-720 (1993).

8. Patrice Y. Simard, Yann LeCun, and John Denker, "Efficient Pattern Recognition Using a New Transformation Distance," *Neural Information Processing Systems* **5** 50-58, Morgan Kaufmann (1993).

9. Léon Bottou and Vladimir Vapnik, "Local Learning Algorithms," *Neural Computation* **4** 888-900 (1992).

10. B. E. Boser, I. Guyon, and V. N. Vapnik, "A Training Algorithm for Optimal Margin Classifiers," in *Proceedings of the Fifth Annual Workshop on Computational Learning Theory* **5** 144-152, Pittsburgh (1992).

11. Corinna Cortes and Vladimir Vapnik, "The Soft Margin Classifier," submitted to *Machine Learning.*

12. Eduard Säckinger and Hans Peter Graf, "A Board System for High-Speed Image Analysis and Neural Networks," Submitted to *IEEE Transactions on Neural Networks.*

13. John S. Denker and Christopher C. J. Burges, "Image Segmentation and Recognition," in **The Mathematics of Induction,** D. H. Wopert (ed.), Addison-Wesley (1994).