

# Distortion-Invariant Recognition via Jittered Queries

Dennis DeCoste and Michael C. Burl

Machine Learning Systems Group  
 Jet Propulsion Laboratory, California Institute of Technology  
 M/S 126-347  
 4800 Oak Grove Drive  
 Pasadena, CA 91109  
 {*decoste, burl*}@aig.jpl.nasa.gov

## Abstract

*This paper presents a new approach for achieving distortion-invariant recognition and classification. A test example to be classified is viewed as a query intended to find similar examples in the training set (or class models derived from the training set). The key idea is that instead of querying with a single pattern, we construct a more robust query, based on the family of patterns formed by distorting the test example. Although query execution is slower than if the invariances were successfully pre-compiled during training, there are significant advantages in several contexts: (i) providing invariances in memory-based learning, (ii) in model selection, where reducing training time at the expense of test time is a desirable trade-off, and (iii) in enabling robust, ad hoc searches based on a single example. Preliminary tests for memory-based learning on the NIST handwritten digit database with a limited set of shearing and translation distortions produced an error rate of 1.35%.*

## 1 Introduction

Achieving invariances to certain types of distortions by providing synthetically-distorted training examples to a learning system is an approach that has met with some success. For example, LeCun supplemented the NIST handwritten digit set with ten random distortions of each training example to encourage his LeNet system to learn invariances to translation, scaling, and skewing [5]. The resulting net produced the lowest error rate (0.7%) reported for the dataset. In the context of support vector machines, the use of “virtual supports”, which are generated post-learning by applying distortions to each support vector, has also shown promise [8].

In this paper we explore a related but novel idea that involves applying a dense set of distortions to

each *test example* at run-time. Since a test example is essentially a query that is intended to find examples or class models that are similar to the probe, this approach can be viewed as a way to generate a more robust query that “understands” how the test example might have looked if it were collected/generated under different circumstances. Rather than burdening the learning system with trying to know how *all* instances of an object class could look under all possible distortions, we focus in on a *particular* instance and ask how that instance could look under the possible distortions.

Shifting the responsibility for handling invariances from the training side to the query side will cause an increase in query execution time, but there are a number of significant advantages. In memory-based learning with a large reference library, e.g., a fingerprint database, expanding the library to include all distortions would be too expensive. Trying to trade space for time by generating distorted versions of each library instance at run-time would also be prohibitive. Note that in our approach, only the test example is distorted, so there is a low space requirement, and the time cost of generating distortions is amortized because the same *jittered query* is made against each instance of the reference library.

One of our primary motivations for query-side jittering is to allow quick model selection during training (moving the invariances to the query side avoids inflating the training time). As an example, consider using cross-validation to select an appropriate distance kernel. In the traditional approach, we would use a training set that has been greatly expanded (through the inclusion of distortions) to learn, for example, a radial basis function (RBF) representation using each of a number of potential kernels. Evaluation of the result-

ing RBF’s over hold-out sets would enable selection of the “best” kernel. With the new approach we would use only the raw training set to learn RBF’s using each potential kernel and then evaluate these RBF’s over the hold-out sets using jittering on the test examples. Instead of multiplying the training time by the number of jitters, the test time, which is typically much shorter, is multiplied. Ultimately, we envision using this procedure to do large-scale model selection via “racing” [6, 2], which will allow quick rejection of inferior hypotheses based on a partial evaluation over the hold-out set.

Model selection could also include cross validation to decide which candidate distortions are useful and worth the time cost. Note that some distortions may actually be harmful, e.g., 180 degree rotations will cause confusion between the digits 6 and 9. Query-side jittering will enable exploration of more candidate distortion sets than would be possible by training with expanded training sets (e.g., using some pre-determined set of distortions). Once a useful set of distortions has been found, the invariances can be “compiled” back onto the model side by relearning on a training set expanded by the useful distortions.

Another scenario, which blurs the distinction between query-side and training-side jittering, involves construction of ad hoc queries from a single example of an object of interest. For example, a user might identify an object in an image and ask an automated system to find similar objects. With just a single example, the system does not have much to go on since there is no knowledge of how the object could vary. However, if the user also specifies a set of (partial) invariances that the object should satisfy, the system can derive a more robust model to use in its search. This idea is discussed in the context of scale invariance in [1]. An added benefit is that the desired invariances are specified at run-time.

Finally, it may be easier to handle large distortions from the query-side rather than the training-side. In particular, a learning algorithm may be incapable of generalizing to large distortions even if it is provided with a large number of training examples. In LeNet5, training examples were all pre-aligned through centering and deslanting. It is not explicitly stated, but inclusion of this step probably indicates that large distortions cannot be handled directly by their learning algorithm.

## 2 Kernel k-Nearest-Neighbors Using Eigen-Digits

For simplicity, this paper explores query-side distortions (i.e. jitters) within the context of k-nearest-

neighbors classification [3].

For generality, we employ nearest-neighbors using the large space of *dot-product kernel* distance metrics, as first suggested in [9]. The distance between two d-dimensional feature vectors  $x_i$  and  $x_j$  is defined as:  $distK(x_i, x_j) = K(x_i, x_i) - 2 * K(x_i, x_j) + K(x_j, x_j)$ , with kernel function  $K(x_i, x_j) = K(x_i^T * x_j)$ . For example, a *polynomial kernel* is defined as  $K(z) = z^p + b$ , for suitable parameters p and b. The same relative distance orderings as standard Euclidian distance (without the final square root) is obtained from  $distK(x_i, x_j)$  using  $p = 1$  and  $b = 0$  for this kernel. Furthermore, using  $p = 2$  gives the Euclidian distance between two expanded vectors of size  $O(d^2)$ , each representing every product of one or two features from the corresponding original vectors  $x_i$  and  $x_j$ . Various values for  $b$  correspond to different relative weightings of each such product. Thus, dot-product kernels provide nonlinearities in a computationally-efficient manner, by squashing dot-products of the original d-dimensional feature vectors — without requiring explicit representation of the underlying large (possibly infinite) expanded dimensionality. Other common kernels include equivalents for 2-layer radial basis and sigmoidal neural networks.

For a d-by-1 query vector  $x_q$  and a d-by-N training set matrix  $B$ , let  $B(:,i)$  indicate the i-th d-by-1 column-vector of  $B$ . Define the corresponding 1-by-N kernel distances  $distK(x_q, B) = [distK(x_q, B(:,1)), \dots, distK(x_q, B(:,N))]$ . This vector  $distK(x_q, B)$  can be obtained simply by first computing: the 1-by-N vector of dot-products  $p_{q,B} = x_q^T * B$ , the scalar  $p_{q,q} = x_q^T * x_q$ , and the 1-by-N  $p_{B,B} = \text{sum}(B .* B)^1$  (Note that  $p_{B,B}$  can be efficiently pre-computed once.) Applying the kernel squashing function  $K(z)$  to each element of those three resulting dot-product vectors, and then combining them according to the definition of kernel distance, yields vector  $distK(x_q, B)$ . Thus, the remainder of this paper focuses on computing (approximating) the dot-product  $p_{q,B}$ . In this paper, the k-nearest-neighbor classification of  $x_q$  is the most-common label of the k examples  $B(:,i)$  with smallest  $distK(x_q, B(:,i))$ .

Brute-force computation of  $p_{q,B}$  corresponds to simple linear scanning of the entire training set, which involves  $O(d*N)$  effort for classifying each query. Standard speedups, such as kd-trees, do not apply well in our context, both because the feature dimensionality (d) of our domains is too high and because the kernel distance metrics are not Euclidian in the origi-

<sup>1</sup>The operator `.*` denotes element-by-element multiplication as in MATLAB. The summation is taken down the columns of  $B .* B$ .

nal feature space. So, instead, we consider techniques similar to those of eigen-faces [12, 13]. Namely, we use Singular Value Decomposition (SVD) to decompose (or approximate)  $B$  into a product  $U_B * S_B * V_B^T$ , where  $U_B, S_B, V_B^T$  are respectively,  $d$ -by- $k_B$ ,  $k_B$ -by- $k_B$ , and  $k_B$ -by- $N$ . For  $k_B < d$ , one can more efficiently (but approximately) compute  $p_{q,B}$  as  $(x_q^T * U_B) * (S_B * V_B^T)$ , with only  $O(d * k_B + k_B^2 * N)$  complexity<sup>2</sup>. One novelty in our approach is that we also do SVD on the query, as will be discussed later.

### 3 Query Jittering

Nearest-neighbor classification should pick the  $k$  training examples with the smallest distance to any jittered version of the query. To obtain some preliminary results on image data, we focussed on the following shearing and translation jitters.

Horizontal *shearing* involves shifting left or right each row of a two-dimensional image by certain discrete numbers of pixels, such that all the shifts corresponding to each row together approximate some slant angle (with zero angle being vertical at the center of the horizontal-axis). In particular, we consider those for which the bottom-most row is shifted in each possible discrete amount. For example, for a 28x28 image, this yields 28 jittered versions (including the original). We mass-recenter the resulting sheared images (so that the average location of the pixel intensities is the center pixel)<sup>3</sup>. Such shearing is a fast approximation that does not change the pixel intensities (only their locations) and is particularly common for domains such as character recognition, where horizontal lines should be retained.

*Box-shift translation* involves shifting the entire image horizontally and/or vertically some discrete amount. A “3x3” box jitter means shifting 0 or 1 pixels along each of the two axes. We perform box jittering on top of each sheared jitter.

#### 3.1 Efficient Jittering via SVD

For a given number  $J$  of jitterings of a given query, the key challenge is to efficiently compute a sufficient approximation of the  $J$ -by- $N$  distance matrix. One technique is to pre-filter: only bother computing distances between query jitters and those (say 1%) of the training examples which are closest to the original query vector. In tangent distance work [11], this is reasonable because they already assume the distance grows smoothly as the query distorts. Query blurring often helps such pre-filtering work well. We have

<sup>2</sup>In practice, to minimize approximation errors, we SVD the training examples for each class separately.

<sup>3</sup>We also assume that each original image is normalized by its standard deviation.

explored the following “query compilation” method which does not require such smoothness.

Let  $A_q$  be the  $J$ -by- $d$  transposed matrix of jitters of query  $x_q$  and  $B$  be the  $d$ -by- $N$  matrix of training data. Brute-force computation of the  $J$ -by- $N$  distance matrix would involve  $O(J * d * N)$  operations. Approximating  $B$  by SVD (retaining  $k_B$  components) reduces time complexity to  $O(J * d * k_B + J * k_B * N)$ . By also approximating  $A_q$  using SVD (retaining  $k_A$  components), one can reduce this complexity to  $O(k_A * d * k_B + k_A * k_B * N + J * k_A * N)$ . This still involves  $O(J * N)$ , but can be  $k_B/k_A$  faster. It can be reasonable, for example, to have  $k_B = 40$  while  $k_A = 4$ , since the query SVD need only minimize variance for related jitters of one example.

To make this work, the SVD of  $A_q$  into  $U_A * S_A * V_A^T$  must be fast. We speed up this step by taking advantage of two facts: we only need  $k_A$  components and  $U_A$  need not be orthogonal for our purposes. This is an ideal application for the recent EMPCA method [7], which has complexity  $O(J * d * k_A)$ . Furthermore, it is an iterative EM algorithm for which a couple of iterations often yields reasonable approximations of  $U_A$  and  $S_A$ . To best approximate  $A_q$ , we take the  $U_A$  and  $S_A$  resulting from a few iterations of EMPCA and then compute  $V_A = S_A^{-1} * U_A^T * A$ .

Ideally, one would determine a good  $k_A$  for each query (e.g., by looking at drop-off in eigenvalues). However, our current implementation assumes a fixed user-supplied  $k_A$ . Nevertheless, the potential to select a query-dependent  $k_A$  is another advantage of doing query-side (vs training-side) jittering.

### 4 Results: MNIST digit recognition

The MNIST data set [4] is a well-studied benchmark that consists of 60,000 training examples and 10,000 test examples. Each example is a 28x28 pixel image with 256 grey-levels and precentered by mass.

Results on the MNIST data set using various query jitters are shown in Figure 1, along with some key results of previous studies.

As in [5], we explored pre-applying shearing distortions to training and test datasets. *Deskewings* refers to replacing an example image with the sheared version for which the first principal axis is as vertical as possible. As expected, the SVD’s of the deskewed training data were somewhat better than for the originals (lower reconstruction errors and faster eigenvalue drop-off, for each class).

The 2.37 vs 2.4 result suggests that using 40 components from SVD of the train set (per class) may give as much benefit as deskewing the datasets, presumably because of the feature-extraction benefits of SVD.

errors	method	deskew	deshear jitter	box jitter	train comps	POLY p
5.0	3-NN				all	1
2.4	3-NN	Y			all	1
2.37	1-NN				40	1
1.92	1-NN	Y	28		40	5
1.92	1-NN	Y			40	1
1.86	1-NN	Y			40	4
1.66	1-NN		28		40	1
1.65	1-NN	Y	28		40	1
1.61	1-NN	Y	28		40	4
1.53	1-NN	Y		3x3	40	1
1.35	1-NN		28	3x3	40	1
1.4	SVM		random (training)		NA	4
1.1	tangent dist		implicit		NA	1
1.1	LeNet4				NA	NA
0.8	virtual SVM		random (training)		NA	9
0.7	boost-LeNet4		random (training)		NA	NA

Figure 1: Errors rates on MNIST 10,000 test examples  
See [4, 5] for details of methods listed in the upper and lower sections of the table.

The 1.66 vs 1.92 result suggests that query-time shear jitters are more effective than deskewing. The 1.35 vs 1.53 result suggests that this holds even when some box jittering is done as well.

Unfortunately, our current implemented class of query jitters (translation and shearing rotation) is a subset of those used in the reported tangent distance result (which includes scaling and line thickness as well). So we do not yet know how much our 1.35 result might improve beyond tangent distance’s 1.1, using all such jitters.

The best reported results involve random distortions to the training set (about 10 per training example). Due to both time and space limitations, systematic distortions of the train set would be infeasible. Virtual support methods for SVM’s attempt to overcome this issue by only distorting supports; however, systematic, densely-sampled distortions will still often be infeasible on the training set side.

We found that using various  $p > 1$  for the polynomial kernel distance has not provided significant test error reduction (contrasting reported support vector machine results). We postulate that perhaps such kernels may be more effective when used in the context of maximum-margin classifiers per se, such as SVM’s. For example, although [10] reports good results using such kernels for nonlinear SVD, the resulting SVD features were ultimately feed into a linear SVM to achieve

those results.

## 5 Conclusions

The related tangent distance approach considers local distortions of both the training and query examples, but only for relatively minor distortions for which smoothness can be assumed. In contrast, our work here tries to avoid making such an assumption, and instead capture local invariances in a query-specific manner, through explicit procedural expansion (jittering) followed by structural compression (SVD).

Even when query jittering is not faster than alternatives for a given query,<sup>4</sup> it might still lead to faster overall training, through use of “racing” and amortizing that cost over many models during model selection. A motivation of this work is that query jittering during racing (using a cheap memory-based method such as nearest-neighbors) might help filter a vast space of alternative distance metrics for a final more-sophisticated memory-based method (such as support vector machines). To make this work well, there would need to be a strong correlation between distance metrics that work well in both methods.

A key limitation of this current work is that ultimately our use of SVD cannot overcome the combinatorial growth in the number of jitters ( $J$ ) for a

<sup>4</sup>For example, our 1.35 MNIST result took 40 hours on a Sun Sparc Ultra60.

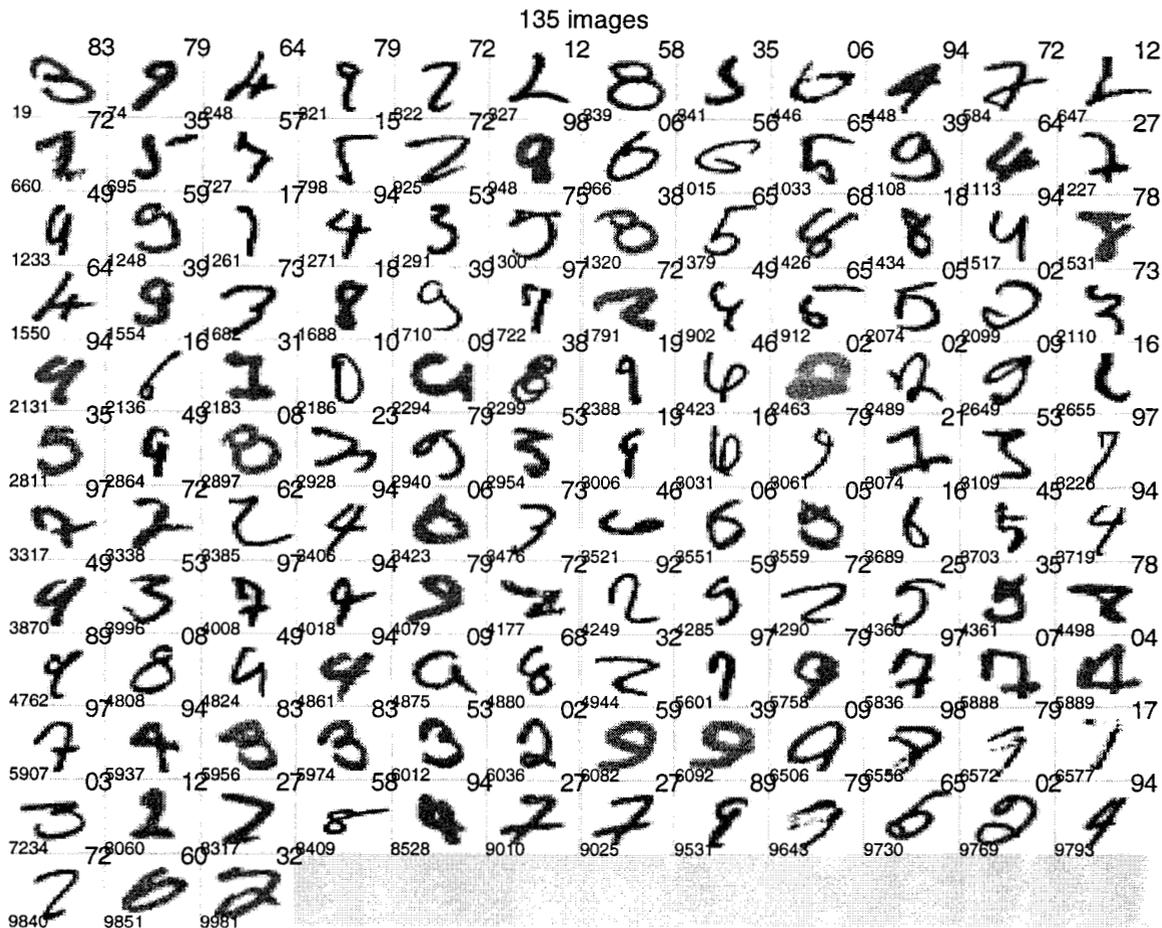


Figure 2: Test errors for 1.35 result

Upper right (of each box): first digit is (mis)classification, second is true label;  
 lower left (of each box): example number (from size 10,000 test set).

given query, which arises from cross-product nature of combining jitter classes. Within our formalism, the challenge is to reduce the effective  $J$  so that the final distance matrix is no longer linear in  $J$  (i.e. avoid constructing the entire  $J$ -by- $N$  matrix). We are currently experimenting with new adaptive methods which do such reduction in the effective  $J$ , by treating each jitter as a training example from which to learn to better interpolate the  $U_A$  mapping matrix over an even larger (implicit)  $J$ .

### Acknowledgments

The research described in this paper was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration.

### References

- [1] M.C. Burl. Continuously-scalable template models. (In review), 1999.
- [2] S. Chien, J. Gratch, and M. Burl. On the efficient allocation of resources for hypothesis evaluation: A statistical approach. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 17(7):652-665, 1995.
- [3] R.O. Duda and P.E. Hart. *Pattern Classification and Scene Analysis*. Wiley, 1973.
- [4] Y. LeCun. MNIST dataset. ([www.research.att.com/~yann/ocr/mnist/](http://www.research.att.com/~yann/ocr/mnist/)).
- [5] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, Nov 1998.
- [6] A. W. Moore and M. S. Lee. Efficient algorithms for minimizing cross validation error. In *Proceedings of the 11th International Conference on Machine Learning*, 1994.

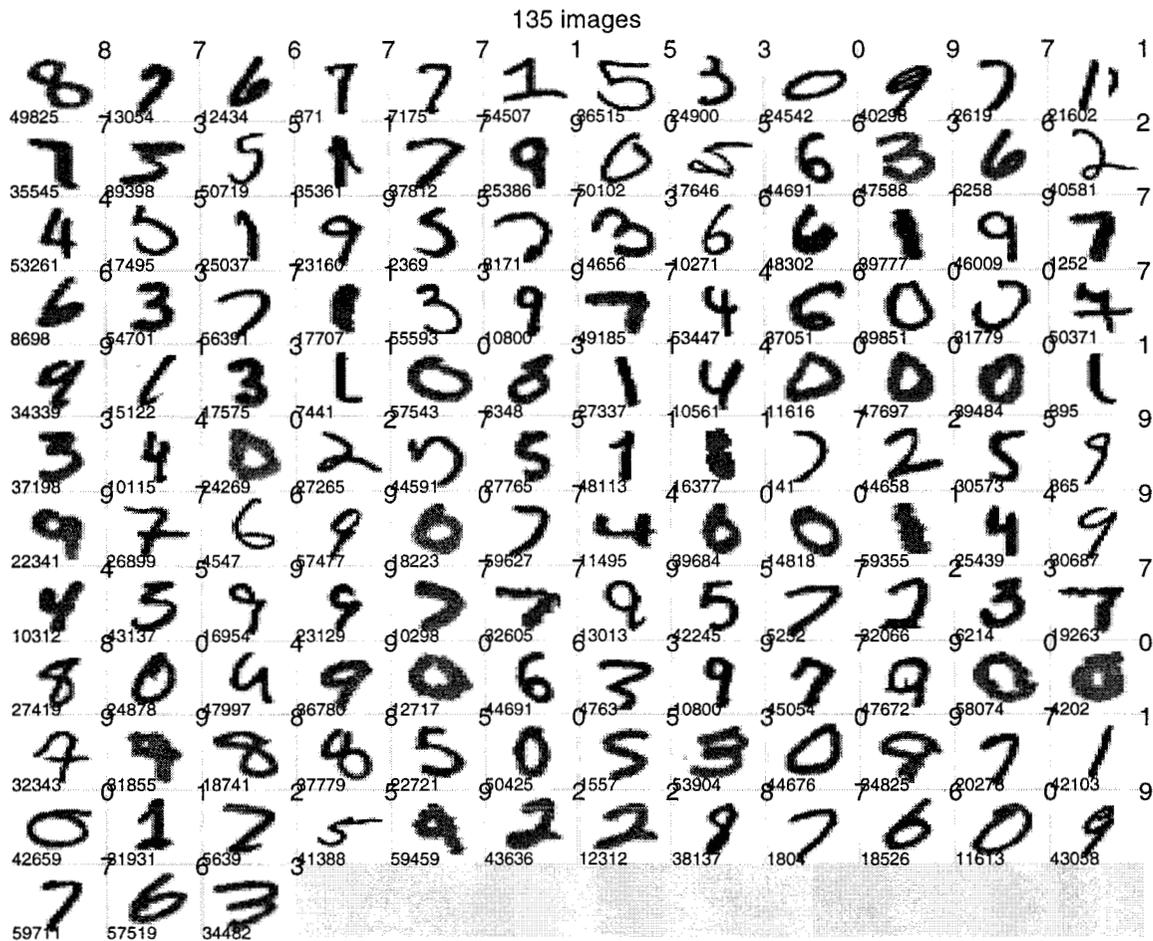


Figure 3: Nearest neighbors for test errors for 1.35 result

Upper right (of each box): true digit label;  
 lower left (of each box): example number (from size 60,000 train set).

[7] Sam Roweis. EM algorithms for PCA and SPCA. In *Neural Information Processing Systems*, volume 10, pages 626–632, 1997.

[8] B. Schölkopf, C. Burges, and V. Vapnik. Incorporating invariances in support vector learning machines. In *Artificial Neural Networks - ICANN'96*, 1996.

[9] B. Schölkopf, A. Smola, and K.R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. Technical report no. 44, Max-Planck-Institut für Biologische Kybernetik, Tübingen, Dec 1996.

[10] B. Schölkopf, A. Smola, and K.R. Müller. Nonlinear component analysis as a kernel eigenvalue problem. *Neural Computation*, 10(5):1299–1319, 1998.

[11] P. Simard, Y. LeCun, and J. Denker. Efficient pattern recognition using a new transformation distance. In *Advances in Neural Information Processing Systems*, volume 5, pages 50–58, 1993.

[12] L. Sirovich and M. Kirby. A low dimensional procedure for the characterization of human faces. *Journal of Optical Society of America*, 4(3):519–524, 1987.

[13] M. Turk and A. Pentland. Eigenfaces for recognition. *Journal of Cognitive Neuroscience*, 3(1), 1991.