Turk J Elec Engin, VOL.9, NO.1 2001, © TÜBİTAK

Combining Multiple Representations for Pen-based Handwritten Digit Recognition

Fevzi ALİMOĞLU, Ethem ALPAYDIN*
Department of Computer Engineering
Boğaziçi University,
TR-80815, İstanbul-TURKEY

Abstract

We investigate techniques to combine multiple representations of a handwritten digit to increase classification accuracy without significantly increasing system complexity or recognition time. In pen-based recognition, the input is the dynamic movement of the pentip over the pressure sensitive tablet. There is also the image formed as a result of this movement. On a real-world database of handwritten digits containing more than 11,000 handwritten digits, we notice that the two multi-layer perceptron (MLP) based classifiers using these representations make errors on different patterns implying that a suitable combination of the two would lead to higher accuracy. We implement and compare voting, mixture of experts, stacking and cascading. Combining the two MLP classifiers we indeed get higher accuracy because the two classifiers/representations fail on different patterns. We especially advocate multistage cascading scheme where the second costlier image-based classifier is employed only in a small percentage of cases.

1. Introduction

Handwritten character recognition has attracted enormous scientific interest due to its evident practical utility. On-line handwriting is important where keyboards are difficult to use, e.g., when the writer is mobile and the device needs to be portable [1]. To achieve high recognition accuracy, many different classification methods have been proposed using statistical, syntactical, or neural approaches. These different methods need not use the same representation but may extract different features to integrate different types of sensors. The usual approach is to choose the one classifier and the particular representation that performs best, discarding the rest. However, it is generally the case that the sets of patterns misclassified by different classifiers do not necessarily overlap, implying that classifiers to a certain extent complement each other. Recently with the cost of computation and memory getting cheaper, it has been possible to build systems that are composed of multiple classifiers [2].

The primary goal of this work is to construct a high-performance system that exploits the difference between two different representations of the same handwritten digit. One representation is dynamic, that

*A previous version of this paper appeared in the Proceedings of the Fourth International Conference on Document Analysis and Recognition, 1997, IEEE Computer Society Press. This work is supported by Tübitak Grant EEEAG-143 and Boğaziçi University Research Funds Grant 00A101D. Francesco Masulli of University of Genova supplied programs for tablet interfacing. Taylan Cemgil solved some of our problems with tablet hardware. Fevzi Alimoğlu is now with the Department of Cognitive and Neural Systems, Boston University, USA.

is, the movement of the pen as the digit is written on a pressure-sensitive tablet. The other representation is static, that is, the image generated as a result of the movement of the pen tip. Two different hand movements for the same character may lead to similar images or different images of the same character may be generated by similar hand movements. Therefore, combining the two representations may improve recognition accuracy. We indeed noticed during our experiments that classifiers using each of the two representations made mistakes for different samples and investigated ways to combine them efficiently.

This paper is organized as follows: In Section 2, we discuss data collection and preprocessing stages for the dynamic and static representations as well as first results with the base classifiers. In Section 3, we advocate the idea of combining multiple classifiers, propose a common notation and discuss in detail and in a comparative manner five different techniques for combining multiple learners. Section 4 contains experimental results on our handwritten digit database and we conclude in Section 5.

2. Data Collection and First Results

We use a WACOM PL-100V pressure sensitive tablet with an integrated LCD display and a cordless stylus attached to the serial port of a PC. The tablet sends x and y tablet coordinates of the pen at fixed time intervals of 100 miliseconds. We collect samples of ten handwritten digits from 44 writers. These writers are asked to write 25 examples from each of the digits in random order inside boxes of 500 by 500 tablet pixel resolution. The raw data captured from the tablet contains enormous variations in writing speed. Various preprocessing methods for on-line handwriting were discussed by Guerfali and Plamondon [3].

From a handwritten digit sample, we generate two feature vectors: The dynamic representation is a sequence of (x_t, y_t) coordinates in time and the static representation is the image formed once the writing of the digit is finished (Figure 1). To get the dynamic representation, the coordinate sequence received from the tablet is first normalized for invariance to translations and scale distortions. It is then spatially resampled to get constant length feature vectors and we get a sequence of points regularly spaced in arc length, as opposed to the input sequence, which is regularly spaced in time. The static representation differs from dynamic representation in the sense that it does not incorporate pen motion. Instead, it is based on solely the image of a character as in off-line recognition tasks. To get the static representation, we first use the scan conversion algorithm to create bitmaps from normalized on-line handwriting data. We then blur and downsample the image to get invariance to small translations and decrease dimensionality. This preprocessing done on the two-dimensional image to get the static representation is costlier than the preprocessing done on the one-dimensional sequence to get the dynamic representation.

After preprocessing, we divide our database from 44 writers into one containing 30 writers and another containing 14 writers. The first part is divided into three sets with 3,748 digits for training (TR), 1,873 for cross-validation (CV) to finetune parameters and 1,873 for writer-dependent testing (WD). 3,498 examples from 14 writers constitute the writer-independent test set (WI) and are used to test performance on writers unseen during training which is the important measure. We have recently donated the database to the UCI Repository under the name `pendigits` and it is available over the internet [4].

We choose multilayer perceptron (MLP) as the base classifier as it is simple and fast once it has been trained. We optimize the feature vector sizes for both dynamic and static representations and finally settle on a sequence of 8 points (16 dimensional vector) and an image size of 8×8 (64 dimensional vector). The results of the two classifiers are given in Table 1.

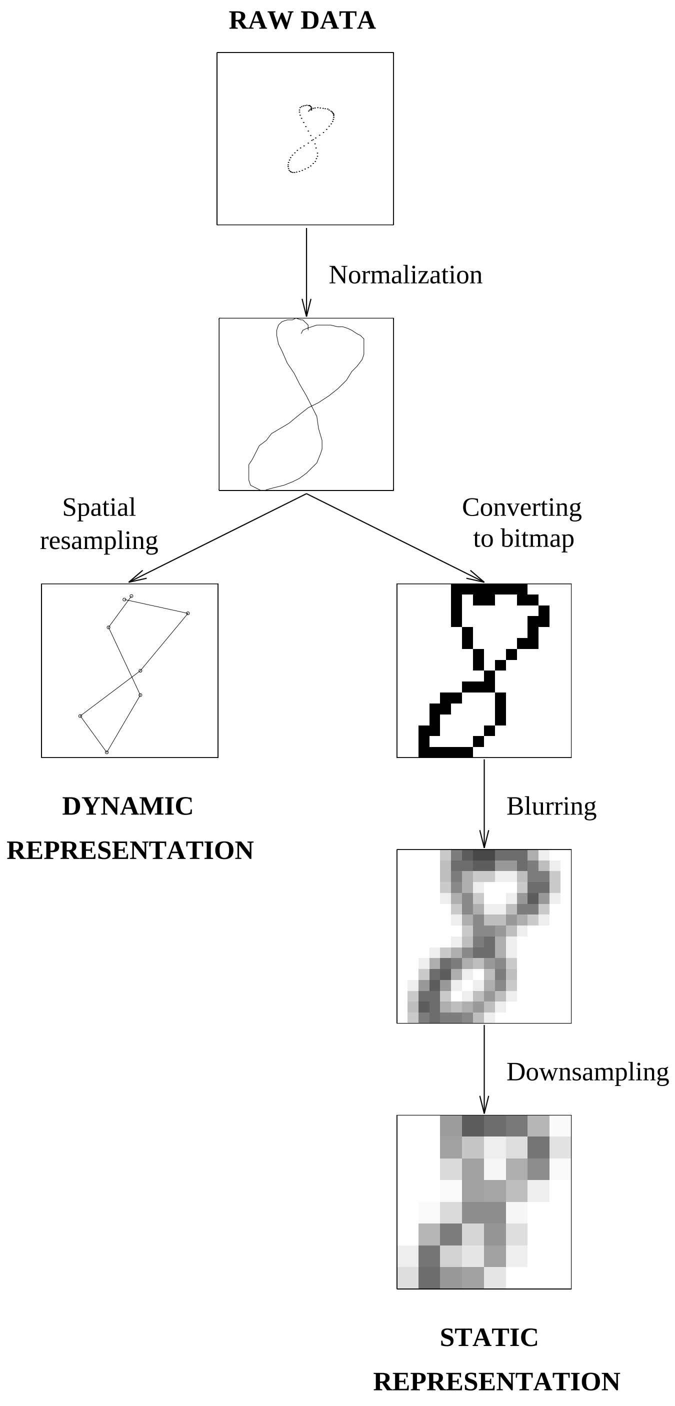

Figure 1. The stages of preprocessing to get the dynamic (pen-movements) and static (image) representations.

We test to see if the two classifiers using the two representations make correlated errors, i.e., fail on different samples or not. In Table 2, error percentages on the sets are given for the two representations. In the same table, error percentage of the cases misclassified by *both* of the classifiers is also given. It can be seen there that though both classifiers have around 5% error on the writer independent test set, if we know which one to use for any pattern, the error percentage can be reduced to less than 2%! Even less error

Table 1. Results with MLP on the two representations. Values are average and standard deviations over ten independent runs. TR: Training set, CV: Cross-validation set, WD: Writer-dependent test set, WI: Writer-independent test set, PS: Number of free parameters, EP: Number of training epochs.

Set	Dynamic MLP	Static MLP
TR	98.98, 0.16	98.76, 0.36
CV	98.33, 0.34	96.24, 0.26
WD	98.26, 0.31	95.73, 0.37
WI	95.26, 0.37	94.25, 0.25
PS	280	760
EP	15.20, 3.29	12.10, 3.90

Table 2. Error percentages of samples misclassified by the two classifiers and by both. These indicate that if we have an ideal selection system to choose which classifier to use, error percentage on WI can be reduced to 1.70% and maybe even less.

Set	Dynamic MLP	Static MLP	Both
TR	1.02	1.24	0.25
CV	1.67	3.76	0.46
WD	1.74	4.27	0.70
WI	4.74	5.75	1.70

may be possible by better integrating the two pieces of information than selecting one of the two classifier outputs.

3. Multiple Classifiers

3.1. Introduction

The performance of a classifier may be finetuned to get the highest possible accuracy on a validation set but this finetuning is a complex task and still there are patterns misclassified even by the best classifier and these patterns may be correctly classified by some other classifier. Because learning is an ill-posed problem, there is no algorithm that can converge to the optimal solution with finite data and each algorithm (or representation) depending on the biases built into it, converges to a different solution and fails under different circumstances. Thus by suitably combining these different alternative solutions, accuracy may be improved. As there is no point in combining multiple classifiers that always make similar decisions, the aim is to be able to find a set of classifiers who differ in their decisions to complement each other.

Combination approaches can be divided into two depending on the representational methodology: In *Uni-Representation*, all classifiers use the same representation. Thus the classifiers themselves should be different to cause them make different decisions. In *Multi-Representation*, different classifiers use different representations of the same input. This is either due to using different sensors or different features extracted from the same input data. In this work, we are interested in multi-representation where there are multiple sources of information and it is desirable to use all of these data to extract more information and achieve higher accuracy in classification. This is similar to *multisensor fusion* where the data from different sensors are integrated.

The simplest approach is *early integration* by *concatenating* all data vectors and treating it as one large vector from a single source. This does not seem theoretically appropriate as this corresponds to modelling data as sampled from one multivariate joint probability distribution. Moreover, larger input dimensionalities make the systems more complex and require larger samples for the estimators to be accurate. The sources also may have different reliabilities which should be taken into account.

The approach we take is *late integration* by making separate decisions using different sources and then combining the decisions. From an algorithmic point of view, there are two possibilities: The classifiers may use the same technique, e.g., MLP. But care must be taken to get different final classifiers. There are different ways by which this can be done: (i) They may use inputs in different representations, i.e., multi-representation, (ii) Training sets may be distinct, e.g., by bootstrapping, (iii) Learning may start from different initial conditions, e.g., in gradient-descent, (iv) Other parameters may be different, e.g., number of hidden units in a MLP, cost function minimized, etc. The different classifiers may also be based on different techniques. For example, one of the classifiers can be statistical and one neural, or one may be programmed as opposed to a trained system.

Orthogonal to these two dimensions, there are two architectural methodologies: In *multiexpert* methods, the classifiers work in parallel. All the classifiers are trained on all patterns and given a pattern, they all give their decisions and a separate combiner computes the final decision using these. Examples are voting (aka committees, ensembles, linear opinion pools), mixture of experts, and stacked generalization. *Multistage* methods use a serial apprach where the next classifier is trained/consulted only for patterns rejected by the previous classifier(s). We advocate multistage methods which suit well to multi-representation where earlier classifiers use simple-to-extract features and handle a majority of the patterns; complex features are only extracted for a small percentage without much increasing the overall complexity. Examples are boosting and cascading.

The gain to be had from combining classifiers depends on how much the classifiers are correlated or dependent. The multiexpert methods assume this whereas the multistage methods actively try to generate such classifiers.

In the present article, we adopt multi-representation where we have two classifiers using the two dynamic and static representations and the approaches that we examine are *Voting*, *Mixture of Experts*, *Stacked Generalization*, *Boosting* and *Cascading*.

3.2. Notation

Let us say that we have m learners and n classes. We denote by d_{ji} the posterior probability estimate of learner j for class i given the input. In this present work, we will consider the case where different d_j use different representations of the input denoted as x_j. The same approaches can also be used in uni-representation if there is only one source in which case all d_j see the same input x. For now we ignore how d_{ji} are calculated and instead we concentrate on the ways to combine them to find r_i the final estimate for class i, given input x.

$$r_i = f(d_{1i}(x_1), d_{2i}(x_2), \ldots, d_{mi}(x_m)|\psi) \tag{1}$$

f is the combining function with ψ denoting its parameters. Assuming that r_i estimates the posterior probability of class i, and that the zero-one loss function is used, to minimize Bayesian risk, we choose the class with the highest probability:

$$i^* = \arg \max_i \ r_i \tag{2}$$

3.3. Voting

Voting is the simplest way to combine multiple classifiers. This is also known as *committees*, *ensembles*, or *linear opinion pools*. It corresponds to taking a linear combination of the learners. First, each classifier is separately and independently trained, e.g., to minimize the cross-entropy between output of learner j on class i, d_{ji} and the desired output y_i, using its own representation x_j over all patterns indexed by p:

$$E_j = -\sum_p \sum_i y_i^p \log d_{ji}(x_j^p), \text{ for all } j. \tag{3}$$

where $y_i^p = 1$ if $x_j^p \in$ class i and 0 otherwise. If we denote by β_j the weight of the vote of the jth classifier, $\psi = \bigcup_{j=1}^m \beta_j$, it is generally required that

$$\forall j, \beta_j \geq 0 \text{ and } \sum_{j=1}^m \beta_j = 1 \tag{4}$$

$$\begin{aligned} r_i &= f_V\left(d_{1i}(x), d_{2i}(x), \ldots, d_{mi}(x) | \beta_1, \beta_2, \ldots, \beta_m\right) & (5)\\ &= \sum_{j=1}^m \beta_j d_{ji}(x_j) & (6) \end{aligned}$$

This is the combination method most worked on. Much research has been done on the uni-representation case and those results carry over to the multi-representation as well. If each voter (classifier) has equal weight $\beta_j = 1/m$ (i.e., we take the average), we call it *simple voting*. It has been shown by Hansen and Salamon [5] that given independent classifiers with success probability higher than 1/2, by taking a majority vote, success increases as the number of voting classifiers increases. Mani [6] has shown that in the case of simple voting, variance decreases as the number of independent voters increase. It has been shown by Jacobs [7] that m dependent experts are worth the same as k independent experts where $k \leq m$. Ali and Pazzani [8] show that there is a substantial correlation between the amount of error reduction due to the use of multiple models and the degree to which the errors made by individual models are correlated.

For the case of multi-representation, Benediktsson and Swain [9] propose to use consensus theory for multisource remote sensing. They discuss choice of weights for linear opinion pools and also the logarithmic opinion pool where weights become exponents and these work better than a statistical classifier using the early-integrated case of concatenated input.

Xu et al. [10] discuss various ways in which the outputs of several classifiers are combined. To compute the weights, they propose to use a *belief* measure or the Dempster-Schafer theory. Perrone [11] gives a number of didactic examples that depict the advantage of voting. He also shows that for minimum square error, when the learners are unbiased and uncorrelated, weights should be inversely proportional to variances. Tumer and Ghosh [12] propose to use order statistics combiners as an alternative to linear combiners that they show empirically to be superior.

Adapting what Perrone [11] stated for regression to classification, if we view each learner as a random noise function added to the true class discriminant function and if these noise functions are uncorrelated with zero mean then the averaging of the individual estimates is like averaging over the noise. In this sense, the voting method is smoothing in the functional space and can be thought of as a regularizer with a smoothness assumption on the true discriminant function.

Kittler et al. [2] analyze different schemes for combining classifiers using different representations by taking the min, product, sum, max, or median of the posterior probabilities computed for each representation and conclude that sum rule is the best. They give two applications to rule identity verification and handwritten digit recognition.

All voting schemes can be seen as approximations under a Bayesian framework with weights approximating prior model probabilities and model decisions approximating model-conditional likelihoods [13] and under certain circumstances, voting and Bayesian techniques will yield identical results [7]. The priors are in turn modeled as distributions with hyperparameters and in the ideal case, one should integrate over the whole model-parameter space. This approach is generally infeasible in practice as modeling the priors and likelihoods analytically is not straightforward; Gibbs sampling is done to get a small number of models with high priors.

It is also possible to use regression to compute the weights during voting by minimizing error on a training set. Lagrange optimization can be used to enforce that weights sum up to one [11]. Or it can be unconstrained and an intercept term may also be added. This is similar to stacking that we discuss in Section 3.5.

3.4. Mixture of Experts

The original mixture of experts (ME) architecture [14, 15] is composed of local experts and there is a gating network whose outputs define the expert weights conditioned on the input. For the multi-representation case, we modify this architecture such that each expert sees one representation and the gating expert sees one of the representations, all representations concatenated (possibly after some suitable dimensionality reduction) or yet another different representation.

$$
\begin{aligned}
r_i &= f_M\left(d_{1i}(x_1), d_{2i}(x_2), \ldots, d_{mi}(x_m) | \psi\right) && (7)\\
&= \sum_{j=1}^{m} \beta_j(x_1, x_2, \ldots, x_m|\psi) d_{ji}(x_j) && (8)
\end{aligned}
$$

where $\beta_j(x|\psi)$ are the gating outputs. This method is like voting except that weights of votes are input dependent, as given by the gating outputs. The training of d_{ji} and β_j are done in a coupled manner to minimize:

$$
E = \log \sum_p \sum_j \beta_j \exp \left[\sum_i y_i^p \log \; d_{ji}(x_j^p) \right] \tag{9}
$$

This error equation forces experts to compete in order to increase the likelihood of the defined normal mixture by preventing overlapping. For a given input, only one β_j is close to one and that expert j is responsible from the giving the system's output. Thus, during training we want to minimize the distance between the required output and the output probabilities of expert j. Gating can thus be seen as implementing a partitioning of the training set among experts which in a way is similar to training experts with subsets of the training set as done by bootstrapping in voting schemes. The difference is that in ME this partitioning is done automatically during learning as a function of locality in the input space and not a priori at random.

3.5. Stacked Generalization

Stacked generalization is a technique proposed by Wolpert [16] that extends voting in that learners (called level 0 generalizers) are not necessarily combined linearly. The combination is made by a combiner system (called level 1 generalizer) that is also trained. We again generalize this architecture for multi-representation where different level 0 generalizers use different input representations.

$$r_i = f_S\left(d_{1i}(x_1), d_{2i}(x_2), \ldots, d_{mi}(x_m)|\psi\right) \tag{10}$$

f_S can be any model and is not restricted to be a weighted sum as in voting or mixture of experts.

The training of d_{ji} and f are done separately. f learns what the correct output is when d_{ji} give a certain output combination. Thus it needs to be trained on data unused in training the d_{ji}. d_{ji} are independently trained as in Eq. (3) and the combiner f_S is trained to minimize on a separate set; using leave-one-out or k-fold cross-validation if a separate set does not exist:

$$E = -\sum_p \sum_i y_i^p \log r_i^p \tag{11}$$

where r_i^p is given as in Eq. (10).

3.6. Boosting

Boosting [17] is a technique for constructing a classifier with small error rates from classifiers which are just slightly better random. Unlike previously described methods, learners are trained serially. After training the first learner, we train the second learner with the data on which the first fail making sure that the two learners complement each other. A third learner is trained with the data on which the first two learners disagree. In the recognition phase, the third learner is consulted only when the first two learners disagree.

In our multi-representation version, the two classifiers use distinct representations. The third classifier may use a third representation, one of the two, or a concatenation of the first two representations.

$$r_i = f_B(c_1, c_2, c_3) \text{ where } c_j = \arg\max_i d_{ji}, j = 1, 2, 3 \tag{12}$$

$$f_B(c_1, c_2, c_3) = \begin{cases} c_1 & \text{if } c_1 = c_2 \\ c_3 & \text{otherwise} \end{cases} \tag{13}$$

Because three separate sets are required to train the three learners, boosting requires large training sets or many runs using k-fold cross-validation and is an expensive method. There is a recent version of boosting called *AdaBoost* [18] which works with smaller datasets and can combine more than three learners but its training strategy is not suitable to the scenario of two distinct representations we have.

3.7. Cascading

The main idea in cascading [19] is that a great percentage of the training samples can be handled by a simple classifier or one that uses a simple feature representation. A complex classifier using a more expensive classification rule or representation is used only for cases that cannot be handled by the simple classifier with enough certainty. Thus the simple classifier handles a majority of the patterns and those that are rejected by it are handled by the complex classifier.

Table 3. Results achieved using different techniques to combine the two pieces of static and dynamic representations.

Technique	TR	CV	WD	WI	PS	EP
Dynamic MLP	98.98, 0.16	98.33, 0.34	98.26, 0.31	95.26, 0.37	280	15.20, 3.29
Static MLP	98.76, 0.36	96.24, 0.26	95.73, 0.37	94.25, 0.25	760	12.10, 3.90
Concat	99.86, 0.06	99.14, 0.10	98.67, 0.27	97.03, 0.41	1,830	13.70, 2.67
Vote	99.55, 0.04	98.94, 0.12	98.41, 0.20	97.09, 0.33	1,040	27.30, 7.19
ME	99.74, 0.11	99.02, 0.27	98.78, 0.25	96.81, 0.36	2,870	19.60, 7.63
Stacking	99.04, 0.13	98.98, 0.14	98.65, 0.25	96.73, 0.14	1,360	129.80, 15.02
Cascading	97.41, 0.42	96.33, 0.48	96.02, 0.57	96.41, 0.52	1,040	115.00, 15.95

The major problem is to decide when a pattern is covered by the simple classifier and thus should be learned by d_S, or is an exception and should be learned by the complex learner d_X; one can of course envisage more than two stages starting from the simplest up to the most complex. We decide on this by looking at the highest class posterior of d_S and comparing it with a threshold θ. If it is greater than θ, we assume that it is covered by the d_S, otherwise it requires more detailed analysis and is handled by d_X. During training, we first train d_S and test it on a separate set and leave aside all patterns that are not covered by d_S; these constitute the training patterns for d_X. During recognition, we first check d_S. If the highest posterior exceeds θ, x is covered by d_S, otherwise it is covered by d_X.

$$f_C(\{d_{Si}\}_i, \{d_{Xi}\}_i) = \begin{cases} c_S = \arg\ \max_i\ d_{Si} & \text{if } \max_i\ d_{Si} > \theta \\ c_X = \arg\ \max_i\ d_{Xi} & \text{otherwise} \end{cases} \tag{14}$$

Pudil et al. [20] take into account costs of measurements and risks in a probabilistic context. They derive conditions in terms of upper bounds of the higher-stage measurements for a multistage classifier to give lower decision risk than a single classifier. Baram [21] shows that partial classification by rejecting doubtful cases produces higher "benefit" values than full classification.

4. Results

For early integration, we combine the two pieces of knowledge by concatenating the two feature vectors to obtain a combined feature vector and train a larger MLP with it. The input vector size is equal to 16+64=80 and leads to a larger MLP. We try different number of hidden units and see that the best is with 20 hidden units. As expected, this has better performance than the individual classifiers (Table 3).

As the simplest way to late integrate, we take a simple vote on the two classifiers. This is as accurate as concatenating and uses less parameters. We use a cooperative mixture of experts architecture where the experts are the two MLP-based classifiers and where the gating is another MLP with 20 hidden units which as input takes a concatenation of the two representations.

In stacked generalization, the level 0 classifiers are the two MLPs. As level 1 generalizer, we use an MLP with 10 hidden units and this performs a little better than a linear combiner (equal to a voting approach with trained unconstrained β_j with an intercept). We use 2-fold cross-validation to train the level 0 and level 1 generalizers which makes this approach longer to train.

We could not implement boosting as we have quite a small number of patterns on which the two MLPs do not agree, which is not sufficient to train a third one. In cascading, we use the dynamic MLP as the simple classifier and the static image-based MLP as the complex one. This is because the preprocessing required for the static MLP is time consuming and cascading is a way to avoid doing this for all patterns. We again use 2-fold cross-validation to train the two classifiers with $\theta = 0.99$ where 30% are rejected by the dynamic MLP and train the static MLP. When θ is less, success on writer dependent set is higher but there are not enough patterns to train the static MLP and thus success on writer independent set is less.

Visual presentation of accuracy versus the number of free parameters for the writer independent test set is given in Figure 2. We note that all combination methods are more accurate than single classifiers on the writer independent test set showing that combining multiple representations does increase accuracy.

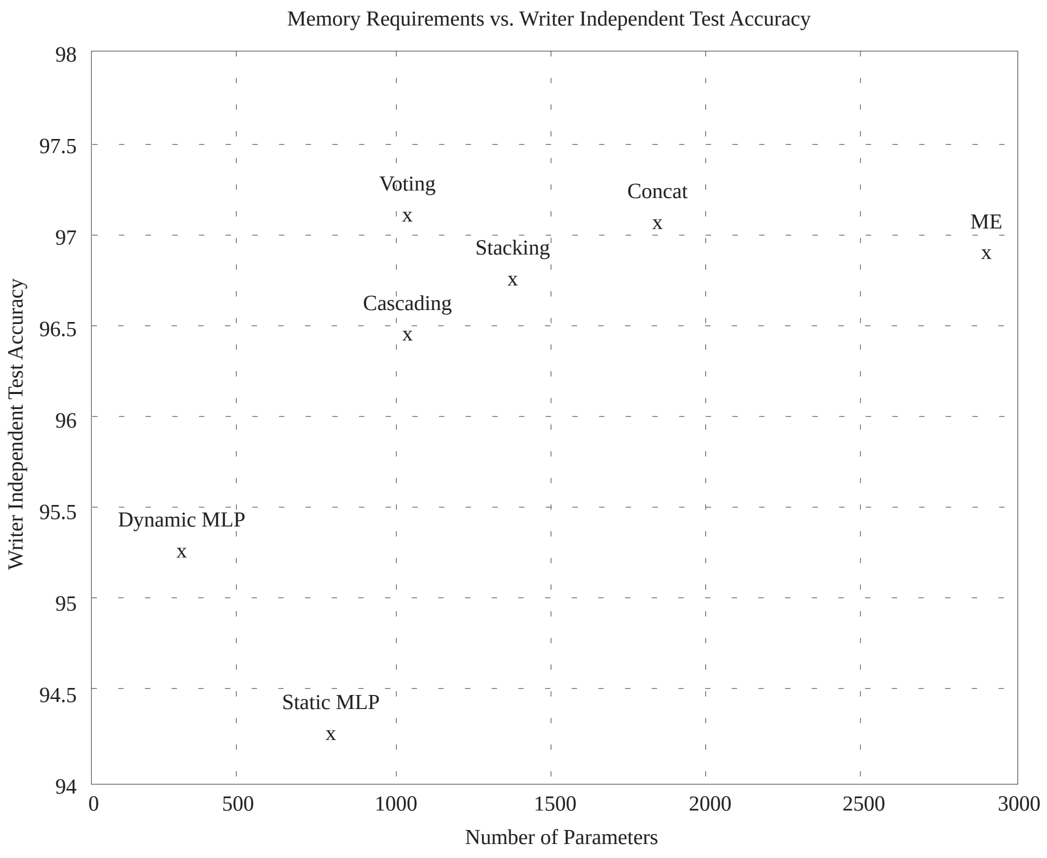

Figure 2. Average accuracy on writer independent test set versus the number of parameters.

Concatenation of the two representations increase accuracy considerably but leads to a large network; the 20 hidden units are completely connected to all 80 inputs. Late integration where there are two classifiers lead to simpler networks, because inputs are smaller. Voting seems to be the best. Stacking is better than voting on writer dependent set but worse on writer independent set, showing that it overfits the training sample; the combination method learned for the 30 writers does not carry over to the 14 left out. Mixture of experts is also successful but needs more parameters as the gating network sees the concatenated input; we also did experiments where the gating network sees the dynamic representation only but those do not generalize as well.

We think that cascading is the most promising approach among all as it leads to the simplest classifier. Preprocessing the static representation is the costly part and by cascading we need this for only 31% of the writer independent test images (14% of the writer dependent test set), leading to the fastest classifier.

5. Conclusions

The accuracy of the MLP classifiers using different representations can be increased by combining them. With the MLP using the dynamic representation, we have around 5% error on the writer independent set. The MLP using the static representation has around 6% error over the same set and seems to be worse than dynamic MLP. Combining the two, error percentage decreases to 3%. Adding a worse classifier decreases the error! This is because the two classifiers/representations fail on different patterns. According to our results, the actual way combining is done is not statistically significant. The choice of the classifiers rather than the choice of combination method is important. The greater the tendency to make correlated errors, the less impressive are the error reductions. The best way to have independent classifiers is to train them on different representations of the input thereby taking into account different data sources that provide complementary data.

In multi-representation, we believe that a multistage method like cascading is more promising as it allows getting rid of the costly preprocessing and feature extraction on some representations. For example in our system, the most expensive processing is in generating the image for the static representation and by cascading, we do not need to compute this for 70% of the test images. Multiexpert methods like voting and stacking always consult all classifiers and thus are slower.

References

[1] Schomaker, L. R. B. "From handwriting analysis to pen-computer applications," *IEE Electronics and Communication Engineering Journal*, Vol. 10, No. 3, pp. 93–102, 1998.

[2] Kittler, J., M. Hatef, R. P. W. Duin, J. Matas. "On combining classifiers," *IEEE Trans. on PAMI*, Vol. 20, No. 3, pp. 226–239, 1998.

[3] Guerfali, W., and R. Plamondon, "Normalizing and Restoring On-line Handwriting," *Pattern Recognition*, Vol. 26, No. 3, pp. 419-431, 1993.

[4] C. J. Merz, P. M. Murphy (1998). UCI Repository of Machine Learning Databases. http://www.ics.uci.edu/∼mlearn/MLRepository.html.

[5] Hansen, L. K., Salamon, P. "Neural Network Ensembles," *IEEE Trans. on PAMI*, Vol. 12, No. 10, pp. 993-1001, 1990.

[6] Mani, G. "Lowering Variance of Decisions by using Artificial Neural Network Ensembles," *Neural Computation*, Vol. 3, pp. 484–486, 1991.

[7] Jacobs, R. A. "Methods for Combining Experts' Probability Assessments," *Neural Computation*, Vol. 7, pp. 867–888, 1995.

[8] Ali, K.M., Pazzani, M.J. *Error Reduction through Learning Multiple Descriptions*, TR 95-39, Dept. of ICS, Univ. of California, Irvine, 1995.

[9] Benediktsson, J. A., Swain, P. H. "Consensus Theoretic Classification Methods," *IEEE Trans. on SMC*, Vol. 22, pp. 688–704, 1992.

[10] Xu, L., Krzyżak, A., Suen, C. Y. "Methods of Combining Multiple Classifiers and Their Applications to Handwriting Recognition," *IEEE Trans. on SMC*, Vol. 22, pp. 418–435, 1992.

[11] Perrone, M. P. *Improving Regression Estimation: Averaging Methods for Variance Reduction with Extensions to General Convex Measure Optimization*, Ph. D. Thesis, Department of Physics, Brown University, 1993.

[12] Tumer, K., Ghosh, J. *Theoretical Foundations of Linear and Order Statistics Combiners for Neural Pattern Classifiers*, Dept. of ECE, Univ. of Texas, Austin, 1995.

[13] Alpaydın, E. "Multiple Networks for Function Learning," *IEEE Intl. Conf. on Neural Networks*, San Francisco: CA, Vol. I, pp. 9–14, March, 1993.

[14] Jacobs, R. A., M. I. Jordan, S. J. Nowlan, and G. E. Hinton, "Adaptive Mixtures of Local Experts," *Neural Computation*, Vol. 3, pp. 79-87, 1991.

[15] Alpaydın, E., and M. I. Jordan, "Local Linear Perceptrons for Classification," *IEEE Trans. on Neural Networks*, Vol. 7, No. 3, pp. 788-792, 1996.

[16] Wolpert, D. H. "Stacked Generalization," *Neural Networks*, Vol. 5, pp. 241–259, 1992.

[17] Drucker, H., Cortes, C., Jackel, L.D., LeCun, Y., and Vapnik, V. "Boosting and Other Ensemble Methods," *Neural Computation*, Vol. 6, No. 6, pp. 1289–1301, 1994.

[18] Freund, Y., and Schapire, R. E. (1996). Experiments with a New Boosting Algorithm. *Proceedings of the Thirteenth International Conference on Machine Learning*, 148–156.

[19] Alpaydın, E., and C. Kaynak, "Cascading Classifiers," *Kybernetika*, Vol. 34, No. 4, pp. 369–374, 1998.

[20] Pudil, P., Novovičová, J., Bláha, S., Kittler, J., "Multistage Pattern Recognition with Reject Option," *11th IAPR International Conference on Pattern Recognition*, Vol. II, IEEE Computer Society Press, 92–95, 1992.

[21] Baram, Y. "Partial Classification: The Benefit of Deferred Decision," *IEEE Trans. on PAMI*, Vol. 20, No. 6, 769–776, 1998.