A trainable feature extractor for handwritten digit recognition

Fabien Lauer [a,*] , Ching Y. Suen [b] and Gérard Bloch [a]

[a] *Université Henri Poincaré – Nancy 1 (UHP), Centre de Recherche en Automatique de Nancy (CRAN UMR CNRS 7039), CRAN–ESSTIN, Rue Jean Lamour, 54519 Vandœuvre Cedex, France*

[b] *Concordia University, Center for Pattern Recognition and Machine Intelligence (CENPARMI), 1455 de Maisonneuve Blvd West, Suite EV003.403, Montréal, QC, Canada, H3G 1M8*

Abstract

This article focusses on the problems of feature extraction and the recognition of handwritten digits. A trainable feature extractor based on the LeNet5 convolutional neural network architecture is introduced to solve the first problem in a black box scheme without prior knowledge on the data. The classification task is performed by Support Vector Machines to enhance the generalization ability of LeNet5. In order to increase the recognition rate, new training samples are generated by affine transformations and elastic distortions. Experiments are performed on the well known MNIST database to validate the method and the results show that the system can outperfom both SVMs and LeNet5 while providing performances comparable to the best performance on this database. Moreover, an analysis of the errors is conducted to discuss possible means of enhancement and their limitations.

Key words: character recognition, support vector machines, convolutional neural networks, feature extraction, elastic distortion

1 Introduction

Handwriting recognition has always been a challenging task in pattern recognition. Many systems and classification algorithms have been proposed in the

* Corresponding author.
Email addresses: fabien.lauer@esstin.uhp-nancy.fr (Fabien Lauer), suen@cenparmi.concordia.ca (Ching Y. Suen), gerard.bloch@esstin.uhp-nancy.fr (Gérard Bloch).

past years. Techniques ranging from statistical methods such as PCA and Fisher discriminant analysis [1] to machine learning like neural networks [2] or support vector machines [3] have been applied to solve this problem.

But since handwriting depends much on the writer and because we do not always write the same character in exactly the same way, building a general recognition system that would recognize any character with good reliability in every application is not possible. Typically, the recognition systems are tailored to specific applications to achieve better performances. In particular, unconstrained handwritten digit recognition has been applied to recognize amounts written on checks for banks or zip codes on envelopes for postal services (the USPS database). In these two cases, good results were obtained. An unconstrained handwritten digit recognition system can be divided into several stages: preprocessing (filtering, segmentation, normalization, thinning...), feature extraction (and selection), classification and verification. This paper focuses on feature extraction and classification.

Since many classifiers cannot process efficiently the raw images or data, feature extraction is a preprocessing step that aims at reducing the dimension of the data while extracting relevant information. The performance of a classifier can rely as much on the quality of the features as on the classifier itself. A good set of features should represent characteristics that are particular for one class and be as invariant as possible to changes within this class. Commonly used features in character recognition are: zoning feature, structural feature, directional features, crossing points and contours. A feature set made to feed a classifier can be a mixture of such features. Besides, to reduce the size of the feature set, feature subset selection can be applied on the extracted features (see for instance [4] for an overview of the methods). In handwriting recognition, features are created from knowledge of the data. But in some other applications, one may not have this knowledge that can be used to develop feature extractors. Another approach to this problem is to consider the feature extractor as a black box model trained to give relevant features as outputs with no prior knowledge on the data. In this scheme, this paper introduces a trainable feature extractor based on the LeNet5 convolutional network architecture.

Amongst all the classifiers that have been applied on character recognition, neural networks became very popular in the 80's as demonstrated by the performances obtained by LeCun's LeNet family of neural networks [5][2]. These are convolutional neural networks that are sensitive to the topological properties of the input (here, the image) whereas simple fully connected networks are not. Another major event in the area was the introduction of Support Vectors Machines (SVM) [3]. The SVMs are based on the Structural Risk Minimization (SRM) principle which minimizes a bound on the generalization risk whereas neural networks aim at minimizing the empirical risk. In the case

of handwritten recognition, SVMs gave very good results [6].

Besides, it has been proved [2] that augmenting the size of the training set helps learning machines to become invariant to certain transformations such as translations. The new samples added to the training set are generated by applying these transformations on the original samples. Then the classifier learns to recognize them as members of the original class. This paper explores the use of the affine transformation and the elastic distortion introduced in [7] to increase the recognition rate in the case of digit recognition.

In the following sections, the article focuses on handwritten digit recognition which is a 10-class problem. The outline of the paper is as follows. Sections 2 and 3 describe two popular classifiers: the convolutional neural network (LeNet5 in particular) and the Support Vector Machine. Then Section 4 exposes the affine and elastic distortions for the expansion of the training set. Section 5 presents a trainable feature extractor based on the LeNet5 architecture and a recognition system that uses SVMs together with this proposed method. Finally, Section 6 exposes the results of the numerical experiments and an analysis of the errors that lead to the conclusion in Section 7.

2 Convolutional neural network

A convolutional neural network is a feed-forwark network that can extract topological properties from an image. It extracts features from the raw image in its first layers and classify the pattern with its last layers. The system is trained like a standard neural network by back propagation. The first layers are usually an alternation of convolutional layers and subsampling layers. A convolutional layer is used to extract elementary visual features from local receptive fields. It is organised in planes, also called feature maps, of simple units called neurons. Each unit has 25 inputs connected to a 5×5 area in the input image (or in the previous layer), which is the local receptive field. A trainable weight is assigned to each connection as for common neural networks, but all units of one feature map share the same weights. This characteristic is justified by the fact that an elementary feature detector useful on one part of the image is likely to be useful on the entire image. Moreover, the weight sharing technique allows to reduce the number of trainable parameters. For instance, LeNet5 has only 60000 trainable parameters out of 345 308 connections. In order to extract different types of local features, a convolutional layer is composed of several feature maps. Because of the local receptive field dimensions, these feature maps have the size of their input minus 4 in both directions (2 pixels loss at each border).

Since the exact location of an extracted feature is irrelevant, a reduction of the

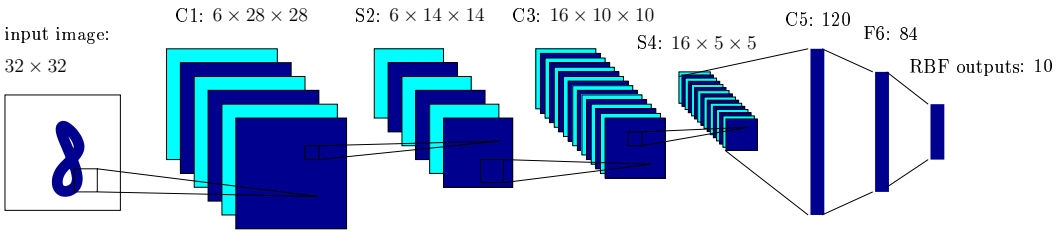

Fig. 1. The architecture of LeNet5. 'C' stands for a convolutional layer, 'S' for a subsampling layer and 'F' for a fully connected layer.

spatial resolution of the feature maps is performed through the subsampling layers. Such a layer comprises as many feature maps as the previous convolutional layer but with half the number of rows and columns. Each unit j is connected to a 2×2 receptive field, computes the average of its four inputs y_i (outputs from the corresponding feature map of the previous layer), multiplies it by a trainable weight w_j and adds a trainable bias b_j to obtain the activity level v_j:

$$v_j = w_j \frac{\sum_{i=1}^{4} y_i}{4} + b_j \tag{1}$$

The weight sharing technique is also applied in the subsampling layers, so that for each feature map, the number of trainable parameters is two (the shared weight and the bias). The description of a particular convolutional neural network known as LeNet5 [2] follows.

LeNet5 takes a raw image of 32×32 pixels as input. It is composed of 7 layers: three convolutional layers (C1, C3 and C5), two subsampling layers (S2 and S4), one fully connected layer (F6) and the output layer. The convolutional and subsampling layers are interlaced as shown in Fig. 1. The first layer is a convolutional layer (C1) composed of 6 feature maps of 28×28 units. The following subsampling layer (S2) reduces by 2 the resolution, while the next convolutional layer (C3) extends the number of feature maps to 16. Here the choice is made not to connect every feature map of S2 to every feature map of C3. Each unit of C3 is connected to several receptive fields at identical locations in a subset of feature maps of S2. These combinations are arbitrary but also reduce the number of free parameters of the network and forces different feature maps to extract different features as they get different inputs. The subsampling layer S4 acts as S2 and reduces the size of the feature maps to 5×5. The last convolutional layer C5 differs from C3 as follows. Each one of its 120 feature maps is connected to a receptive field on all feature maps of S4. And since the feature maps of S4 are of size 5×5, the size of the feature maps of C5 is 1×1. Thus C5 is equivalent to a fully connected layer. It is still labeled as a convolutional layer because if the input image was larger, the dimension of the feature maps would be larger. The fully connected layer (F6) contains 84 units connected to the 120 units of C5.

All the units of the layers up to F6 have a sigmoidal activation function φ of

the type:

$$y_j = \varphi(v_j) = A \tanh(S v_j) \tag{2}$$

where v_j is the activity level of the unit. A and S are two constant parameters for the sigmoid function (see [2] for the setting of A and S). Finally, the output layer is an Euclidean RBF layer of 10 units (for the 10 classes) whose outputs y_j are computed by:

$$y_j = \sum_{i=1}^{84} (y_i - w_{ij})^2 \qquad j = 0, \ldots, 9 \tag{3}$$

where y_i is the output of the ith unit of the layer F6. For each RBF neuron, y_j is a penalty term measuring the fitness of its inputs y_i to its parameters w_{ij}. These parameters are fixed and initialized to -1 or $+1$ to represent stylised images of the characters drawn on a 7×12 bitmap that are targets for the previous layer (hence the size 84 for the layer F6). Then the minimum output gives the class of the input pattern.

3 Support Vector Machines

Support vector machines (SVMs) were introduced in [3] as learning machines with capacity control for regression and binary classification problems. In the case of classification, a SVM constructs an *optimal separating hyperplane* in a high-dimensional feature space. The computation of this hyperplane relies on the maximization of the margin.

3.1 Optimal separating hyperplane

The optimal separating hyperplane is a margin classifier whose output is given by:

$$f(x) = \text{sign } \left(x^T w + b \right) \tag{4}$$

with the input pattern x and where the bias b and the vector of weights w are trained by maximizing the margin $1/\|w\|$ under the constraint that the N training patterns are well classified and outside the margin:

$$\min \frac{1}{2} \|w\|^2$$

$$\text{subject to } y_i(x_i^T w + b) \geq 1, \quad i = 1, \ldots, N$$

with $y_i \in \{-1, 1\}$ representing the label of the training pattern x_i.
The solution corresponds to the saddle point of the primal Lagrangian:

$$L_P = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{N} \alpha_i[(y_i(x_i^T w + b) - 1] \tag{5}$$

where the α_i are the Lagrange multipliers.
This problem leads to the maximization of the dual Lagrangian with respect to α_i:

$$\begin{aligned} \max L_D &= \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i,j}^{N} \alpha_i \alpha_j y_i y_j (x_i^T x_j) \\ \text{subject to} \quad \alpha_i &\geq 0, \qquad i = 1, \ldots, N \\ \textstyle\sum_{i=1}^{N} \alpha_i y_i &= 0 \end{aligned} \tag{6}$$

The resulting separating rule is:

$$f(x) = \text{sign}\left(\sum_{support\ vectors} y_i \alpha_i (x_i^T x) + b\right) \tag{7}$$

where the x_i are the support vectors (SVs) with non-zero corresponding Lagrange multipliers α_i. The SVs are the training patterns that lie on the margin boundaries. An advantage of this algorithm is its sparsity since only a small subset of the training examples are used to compute the output of the classifier.

3.2 Soft margin SVM

The soft margin separating hyperplane is used to deal with non-separable data. A set of slack variables ξ_i is introduced to allow errors (or points inside the margin) during the training. A hyperparameter C is used to tune the trade-off between the amount of accepted errors and the maximization of the margin:

$$\begin{aligned} &\min \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{N} \xi_i \\ &\text{subject to } y_i(x_i^T w + b) \geq 1 - \xi_i \quad i = 1, \ldots, N \end{aligned}$$

This new formulation leads to the same dual problem but with *box constraints* on the Lagrange multipliers:

$$0 \leq \alpha_i \leq C \qquad i = 1 \ldots N \tag{8}$$

The tuning of the hyperparameter C is a delicate task. A common method is to perform a grid search, i.e. to test many values of C and estimate for

each the generalization error (usually by cross-validation or on an independent validation set). But this procedure is very time consuming. Some authors proposed other methods including evolutionary tuning [8] or gradient-based approaches [9][10] for tuning of SVM hyperparameters.

3.3 Nonlinear mapping and the kernel function

As it can be seen in (6) and (7), the input vectors are only involved through their inner product. Thus, to map the data in a feature space, one does not need to consider the feature space in explicit form. One only has to calculate the inner products of the vectors in the feature space via the kernel function $K(\cdot, \cdot)$. This is the *kernel trick* that allows the construction of a decision function that is nonlinear in the input space but equivalent to a linear decision function in the feature space:

$$f(x) = \text{sign} \left(\sum_{support\ vectors} y_i \alpha_i K(x_i, x) + b \right) \tag{9}$$

where $K(x_i, x)$ stands for the kernel function. Typical kernel functions are:

- RBF kernel: $K(x_i, x) = \exp\left(\frac{-\|x - x_i\|^2}{2\sigma^2}\right)$
- Sigmoid kernel: $K(x_i, x) = \tanh\left(\gamma(x^T x_i) + c\right)$
- Polynomial kernel: $K(x_i, x) = (\gamma x^T x_i + c)^d$

3.4 Multi-class SVM

There are two common methods to solve a multi-class problem with binary classifiers such as SVMs: one-against-all (or one-vs-rest) and one-against-one. In the one-against-all scheme, a classifier is built for each class and assigned to the separation of this class from the others. For the one-against-one method, a classifier is built for every pair of classes to separate the classes two by two. Another approach to the recognition of n different digits is to use a single n-class SVM instead of n binary SVM subclassifiers with the one-against-all method, thus solving a single constrained optimization problem. Multi-class SVMs have been studied by different authors [11], [12] and [13]. But this method is not very popular in digit recognition applications yet and did not yield better performances than other classifiers. In [14], a multi-class SVM was compared to a group of binary SVMs on the USPS datasets. The multi-class SVM gave lower accuracy rates than the common methods. However, multi-class SVMs gave promising results and outperformed other combinatory methods in the prediction of protein secondary structures [15].

4 Expanding the training set

If the number of training samples is small and if the distribution to be learned has some transformation-invariance properties, generating additional data using transformations may improve the recognition performances [7]. In [2], results on the MNIST database were improved by applying affine transformations on the data and thus multiplying the size of the training set by ten. Thus one can create new training samples by using prior knowledge on transformation-invariance properties in order to improve the learning of a machine.

4.1 Affine transformation

Simple distortions such as translations, rotations, scaling and skewing can be generated by applying affine displacement fields to images. For each pixel (x, y), a new target location is computed with respect to the displacement field $\Delta x(x, y)$ and $\Delta y(x, y)$ at this position. For instance if $\Delta x(x, y) = \alpha x$ and $\Delta y(x, y) = \alpha y$, the image is scaled by α. In case α is a non-integer value, then bilinear interpolation is performed.

The general form of affine transformation is:

$$\begin{pmatrix} x \\ y \end{pmatrix} = A \begin{pmatrix} x \\ y \end{pmatrix} + B + \begin{pmatrix} x \\ y \end{pmatrix} \tag{10}$$

where the 2×2-matrix A and the column-vector B are the parameters of the transformation. For instance, for scaling:

$$A = \begin{pmatrix} \alpha & 0 \\ 0 & \alpha \end{pmatrix}, \qquad B = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \tag{11}$$

In this paper, translations of one pixel in the 8 directions are used to generate new samples and multiply the size of the training set by 9. For these transformations, $A = 0$ and B takes 8 different values, in the 4 main directions:

$$\begin{pmatrix} 1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \begin{pmatrix} -1 \\ 0 \end{pmatrix}, \begin{pmatrix} 0 \\ -1 \end{pmatrix} \tag{12}$$

and in the 4 diagonals:

$$\begin{pmatrix}1\\1\end{pmatrix}, \begin{pmatrix}-1\\1\end{pmatrix}, \begin{pmatrix}-1\\-1\end{pmatrix}, \begin{pmatrix}1\\-1\end{pmatrix} \tag{13}$$

4.2 Bilinear interpolation

Bilinear interpolation is used to evaluate the value of a transformed pixel whose coordinates are not integers. After a displacement, a pixel is surrounded by a square of pixels. The new pixel value is computed from the values of the pixels ($p_i,\ i = 1, \ldots, 4$) at the corners (bottom left, bottom right, top left and top right) of the original image as follows.

Let (x, y) be the original coordinates of pixel p, (u, v) the new ones, p_{i_x} and p_{i_y} the coordinates of the corner p_i. First, two interpolations are made horizontally, with respect to the relative new coordinates inside the square $(u_r = u - p_{1_x}, v_r = v - p_{3_y})$:

$$\begin{aligned} pi_{Xbottom} &= p_1 + u_r(p_2 - p_1) \\ pi_{Xtop} &= p_3 + u_r(p_4 - p_3) \end{aligned} \tag{14}$$

Finally the vertical interpolation is performed on the previous results:

$$p = pi_{Xbottom} + v_r(pi_{Xtop} - pi_{Xbottom}) \tag{15}$$

4.3 Elastic distortion

Elastic distortion is an image transformation introduced by [7] to imitate the variations of the handwriting to create new data and increase the performances.

The generation of the elastic distortion is as follows. First, random displacement fields are created from a uniform distribution between -1 and $+1$. They are then convolved with a Gaussian of standard deviation σ. After normalization and multiplication by a scaling factor α that controls the intensity of the deformation, they are applied on the image. σ stands for the elastic coefficient. A small σ means more elastic distortion. For a large σ, the deformation approaches affine, and if σ is very large, then the displacements become translations.

In the proposed method, the elastic distortions are applied on every sample

Fig. 2. Samples generated by elastic distortion from the orginal pattern shown on the left.

of the training set to generate 9 new samples for each one, thus multiplying the size of the training set by 10. Figure 2 shows some samples generated by elastic distortion.

5 Feature extraction

In handwriting recognition, features and feature extractors are created from some knowledge of the data. A feature extractor processes the raw data (the gray-scaled image in this case) to generate a feature vector. This vector has a smaller dimension than the orginal data while holding the maximum amount of useful information given by the data. As an example, a feature extractor might build a feature vector whose component i is the number of crossing points in the ith line of the image. This feature extractor is constructed from prior knowledge on the application, because we know that the crossing points possess pertinent information for differentiating digits.

5.1 Trainable feature extractor

Another approach to this problem is to consider the feature extractor as a black box model trained to give relevant features as outputs with no prior knowledge on the data. As explained in the Section 2, LeNet5 extracts the features in its first layers. The weights of these layers are trained so that its output layers can minimize the classification error. The idea is to replace the last two layers by a single linear output layer. For a 10-class problem, this layer has 10 units, each one performing a linear combination of the previous layer outputs and giving a confidence measure for a class. Thus the 120 outputs of the last convolutional layer are trained to be linearly separable and can be used as features for any other classifier once the network has been trained. The resulting system is a trainable feature extractor (TFE) that can quickly be applied to a particular image recognition application without prior knowledge on the features. Figure 3 shows its architecture in the case of digit recognition with 10 classes.

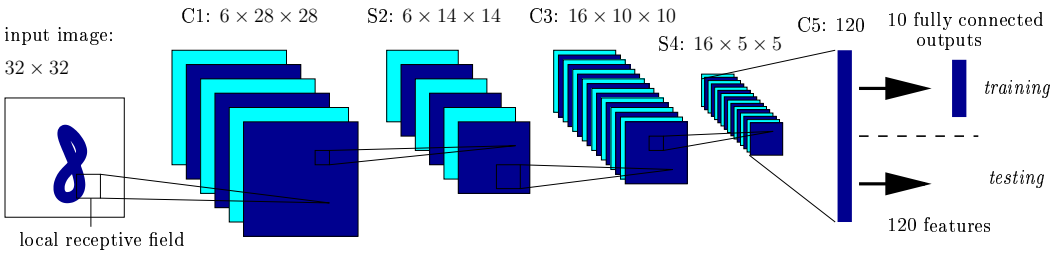

Fig. 3. The architecture of the trainable feature extractor (TFE). The outputs of the layer C5 are either directed to the 10 outputs for the training phase or used as features for the testing phase.

5.2 The digit recognition system based on the TFE

The proposed system is composed of the trainable feature extractor (TFE) of Fig. 3 connected to a multitude of binary SVMs for the testing phase. It is thus labeled TFE-SVM in the following experiments. The multi-class approach for the SVMs is either the one-vs-one method that needs 44 binary classifiers to separate every couples of classes or the one-vs-rest method that involves 10 classifiers, each one assigned to the separation of one class from the others. In the following experiments, both methods are applied and the Tables show only the best of the two results.

The idea of using the extracted features of a convolutional network can be found in [16]. The last layers of a LeNet4 network were replaced by a K-NN classifier to work on the extracted features. But this method did not improve the results compared to a plain LeNet4 network.

6 Experiments

This section presents the results obtained by the proposed methods on the MNIST database [17]. This database is a widely known benchmark that contains a training set of 60000 images and a test set of 10000 images. The images are gray-level bitmaps normalized and centered in a 28×28 pixels frame. The efficiency of the trainable feature extractor is first verified together with the use of SVMs instead of standard LeNet5 outputs. The gain in recognition rate provided by the expansion of the training set is then evaluated and a comparison between the elastic distortion and affine transformations is performed.

Table 1
Training error rates of SVMs used with trained features and various kernels. *poly5* stands for a polynomial kernel of degree 5.

kernel	training set size	training error (%)
linear	15000	0.02
poly5	15000	0
linear	30000	0.30
poly5	30000	0
linear	60000	0.19
poly5	60000	0.90
rbf	60000	0.51

6.1 Trainable feature extractor and SVMs

First, the relevance of the trained features is verified. Table 1 presents the error rates on training sets of different sizes for SVMs that use the trained features with various kernels. It shows that the features provided by the trainable feature extractor are relevant to the task and almost linearly separable.

The gain in generalization obtained by replacing the outputs of the convolutional neural network by SVMs, as in the TFE-SVM scheme, is then estimated by a comparison between a full network (LeNet5 with a simple output layer) with its own outputs and the same network with SVM outputs. The SVM hyperparameters (C and σ) are tuned by cross validation (5-fold) on a small subset of the training set (15000 samples). The results presented in Table 2 show that the SVMs perform better than the network outputs on the test set even though the features are not trained for the SVMs but for these outputs. This set of experiments validates the proposed recognition system labeled TFE-SVM based on the idea of connecting SVMs to a convolutional neural network after the training phase.

6.2 Expanded training set

A comparison between the affine transformations and the elastic distortions for the expansion of the training set is conducted. The affine transformations create 8 new samples for each sample of the training set by translations of one pixel in the 8 directions. The parameters for the elastic distortion are $\sigma = 14$ and $\alpha = 1$, the size of the training set being multiplied by 10. Table 3 exposes the recognition rate on the test set of LeNet5 trained either on translated or

Table 2
Comparison of the recognition rate on the test set between a full neural network (NN) and SVM classifiers using the outputs of the last convolutional layer of the same network as input features.

Classifier	training set size	test rec. rate
NN	15000	98.45
SVM poly5	15000	98.57
NN	30000	98.63
SVM poly5	30000	98.86
NN	60000	98.70
SVM linear	60000	98.94
SVM poly5	60000	98.96
SVM rbf	60000	99.17

elasticly distorted samples. This Table proves that generating new samples for

Table 3
Comparison between the elastic distortions and the affine transformations (translations) for the expansion of the training set. The lines *none* correspond to the original training set before the generation of new samples.

distortion	training set size	test rec. rate
none	15000	98.45
elastic	150000	**98.99**
affine	135000	98.95
none	30000	98.63
elastic	300000	**99.22**
affine	270000	99.15
none	60000	98.70
elastic	600000	99.28
affine	540000	**99.32**

the training set based on some knowledge on transformation-invariance properties can increase the recognition rate. In all these experiments, a gain of at least 0.5% to the performances is obtained for both transformations. Actually, the choice to be made between the elastic distortion and the affine transformations is not clear. The elastic distortion appears to perform better on small subsets of the training set, while the translations offer the best performance on the whole dataset. Nonetheless, care must be taken not to generate too many samples. Tests showed that multiplying the size of the training set by

20 with elastic distortions lead to lower results.

6.3 TFE-SVM and distortions

Now, the proposed system (TFE-SVM) is used in combination with the affine transformation and the elastic distortion for the training set expansion. The same tranformations as in the previous section are applied. The SVMs have RBF kernels and the hyperparameters are set to the same values as in the previous experiments.

Table 4 shows published results of different methods on the MNIST database together with the best results of this work. Firstly, the upper part of the Table shows that the proposed method outperforms both LeNet5 and SVMs without the use of new samples to expand the training set. Moreover, when the affine transformations are applied, a recognition rate of 99.46% (error rate of 0.54%) is obtained which is very close to the best performance of 99.60% achieved on the MNIST database to this day. In [7], this performance was obtained with a convolutional neural network (with a different architecture than LeNet5) used together with the elastic distortion.

6.4 Analysis of the errors

This section focusses the errors on the test set of the TFE-SVM trained on the training set extended by affine transforamtions. The 54 misclassified patterns out of the 10000 test samples are shown in Fig. 4.

The confusion matrix (Table 5) allows to see the frequency and the types of the errors (which digit is mistaken for which). It shows that the most difficult digits to classify are '4','6' and '8', and that most frequent confusing pairs are '4-6','5-3' and '9-4'. Besides, many of the indexes of errors are the same as those of LeNet5 given in [20]. In this latter study, about 2/3 of the errors were categorized as easily recognized by humans without any ambiguity. Looking at the patterns in Fig. 4, 32 samples out of 54 (59%) could be considered as recognizable by humans without any ambiguity. The remaining samples are either too distorted, broken or ambiguous. So the system can still be enhanced to increase the performance but only up to a certain limit which would be 0.22% error rate, so a recognition rate of 99.78%. If we remove these 22 bad samples from the test set and compute the error rate of the system on recognizable samples only, then the TFE-SVM achieves an error rate of only 0.32%.

From the 32 remaining errors, 12 are due to the preprocessing (scanning, nor-

Table 4
Published results of other methods on the MNIST database. The *distortion* column indicates the type of distortion used to expand the training set.

Classifier	distortion	reference	test error (%)
SVM		[18]	1.4
LeNet5		[2]	0.95
TFE-SVM		**this paper**	**0.83**
VSVM	affine	[18]	0.8
LeNet5	affine	[2]	0.8
LeNet5	**elastic**	**this paper**	**0.72**
boosted-LeNet4	affine	[2]	0.7
LeNet5	**affine**	**this paper**	**0.68**
VSVM2	affine	[18]	0.68
K-NN		[19]	0.63
VSVM2+deskewing	affine	[18]	0.56
TFE-SVM	**elastic**	**this paper**	**0.56**
TFE-SVM	**affine**	**this paper**	**0.54**
convolutional NN	elastic	[7]	0.4

malization, orientation...) of the creation of the database. These samples are very distorted or rotated but are still recognizable by humans. Some of these samples also have broken or overlapping strokes because of the thickness of the pen or cursive writing. Other errors come from patterns of strange prototypes, that are written in atypical style, such as the '4's found in Fig. 4. Thus 8 samples can be considered as misclassified because of the lack of samples of the same prototype in the training set. Though they are perfectly clear, the classifier has not learned to recognize them. This problem could be tackled by generating more samples of these prototypes to extend the training set. In this scheme the training set should be first classified in terms of prototypes within the classes. Another classifier is thus required to extract the patterns of rare prototypes (see for instance [21] for a study of the digit prototypes and their classification).

7 Conclusion

This paper presented two of the best classifiers commonly used in handwriting recognition: the LeNet5 convolutional neural network and the Support Vec-

Fig. 4. The 54 patterns of the test set misclassified by the TFE-SVM trained on the training set extended by affine transformations. The labels appear above the image (`target->output`) and the sample index below.

tor Machine (SVM). The problem of feature extraction has been tackled in a black box approach and a trainable feature extractor based on LeNet5 architecture has been introduced. The proposed method (TFE-SVM) has been tested on a well known benchmark for handwritten digit recognition, the MNIST database. The TFE-SVM outperformed both LeNet5 and SVMs trained on the original training set. Used together with the elastic distortion or affine transformations for the expansion of the training set, the proposed method achieved performances comparable to the best published results on this database. This shows the efficiency of generating new samples, and that simple translations can be as good as elastic distortions for this purpose. Moreover, the trainable feature extractor allowed to obtain these results without prior knowledge on the construction of features for handwritten character recognition. This

Table 5
Confusion matrix on the test set for the TFE-SVM with affine transformations for the training set expansion.

target / output	0	1	2	3	4	5	6	7	8	9
0			1				1		1	
1							2	2		
2		1		1			1	3	2	
3						4			1	
4							1			4
5				1			3		2	1
6	1	1			5	1			1	
7			3		1				1	
8	1	1	1							1
9					2			1	1	
Total	2	3	5	2	8	5	8	6	9	6

is the main advantage of the method that can be quickly applied to image recognition applications for which little is known on relevant features.

An analysis of the errors allowed to conclude that the performance could not be increased above a certain limit, because of bad samples in the test set not recognizable without ambiguity by humans. Nonetheless some error patterns are very clear and are still misrecognized because of their structure and the lack of samples of the same prototype in the training set. Regarding these errors, the generation of more samples of rare prototypes could lead to further improvement.

Future studies might consider using the architecture of the convolutional network presented in [7], that gave the best result on the MNIST database, for the TFE instead of LeNet5. Together with the use of the cross-entropy criterion for the training of the network this architecture might provide better features for the SVM classifiers.

References

[1] R. O. Duda, P. E. Hart, D. G. Stork, Pattern Classification, 2nd Edition, Wiley, New York, 2000.

[2] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, Proc. of the IEEE 86 (11) (1998) 2278–2324.

[3] V. N. Vapnik, The Nature of Statistical Learning Theory, Springer-Verlag, New York, 1995.

[4] I. Guyon, A. Elisseeff, An introduction to variable and feature selection, Journal of Machine Learning Research 3 (2003) 1157–1182.

[5] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel, Handwritten digit recognition with a back-propagation network, in: D. Touretzky (Ed.), Advances in Neural Information Processing Systems, Vol. 2, M. Kaufman, Denver, CO, 1990, pp. 396–404.

[6] C. J. C. Burges, B. Schölkopf, Improving the accuracy and speed of support vector machines, in: M. Mozer, M. Jordan, T. Petsche (Eds.), Advances in Neural Information Processing Systems, Vol. 9, The MIT Press, 1997, pp. 375–381.

[7] P. Y. Simard, D. Steinkraus, J. C. Platt, Best practices for convolutional neural networks applied to visual document analysis, in: Proc. of the 7th Int. Conf. on Document Analysis and Recognition, Vol. 2, Edinburgh, Scotland, 2003, pp. 958–962.

[8] F. Friedrichs, C. Igel, Evolutionary tuning of multiple SVM parameters, in: M. Verleysen (Ed.), Proc. of the 12th Europ. Symp. on Artificial Neural Networks (ESANN), Bruges, Belgium, 2004, pp. 519–524.

[9] O. Chapelle, V. N. Vapnik, O. Bousquet, S. Mukherjee, Choosing multiple parameters for support vector machines, Machine Learning 46 (1-3) (2002) 131–159.

[10] N. E. Ayat, M. Cheriet, C. Y. Suen, Optimizing the SVM kernels using an empirical error minimization scheme, in: S. Lee, A. Verri (Eds.), Pattern Recognition with Support Vector Machines. Lecture Notes in Computer Science, Vol. 2388, 2002, pp. 354–369.

[11] J. Weston, C. Watkins, Multiclass support vector machines, Tech. Rep. CSD-TR-98-04, Royal Holloway, University of London (1998).

[12] K. Crammer, Y. Singer, On the algorithmic implementation of multiclass kernel-based vector machines, Journal of Machine Learning Research 2 (2001) 265–292.

[13] Y. Guermeur, A. Elisseeff, H. Paugam-Moisy, A new multi-class SVM based on a uniform convergence result, in: Proc. of the Int. Joint Conf. on Neural Networks, Vol. 4, 2000, pp. 183–188.

[14] E. J. Bredensteiner, K. P. Bennett, Multicategory classification by support vector machines, Computational Optimization and Applications 12 (1999) 53–79.

[15] Y. Guermeur, Combining discriminant models with new multi-class support vector machines, Pattern Analysis and Applications 5 (2) (2002) 168–179.

[16] Y. LeCun, L. D. Jackel, L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, U. A. Muller, E. Sackinger, P. Simard, V. Vapnik, Learning algorithms for classification: A comparison on handwritten digit recognition, in: J. H. Oh, C. Kwon, S. Cho (Eds.), Neural Networks: The Statistical Mechanics Perspective, World Scientific, 1995, pp. 261–276.

[17] Y. LeCun, The MNIST database of handwritten digits, http://yann.lecun.com/exdb/mnist/index.html.

[18] D. Decoste, B. Schölkopf, Training invariant support vector machines, Machine Learning 46 (2002) 161–190.

[19] S. Belongie, J. Malik, J. Puzicha, Shape matching and object recognition using shape contexts, IEEE Trans. on Pattern Analysis and Machine Intelligence 24 (2002) 509–522.

[20] C. Y. Suen, J. Tan, Analysis of errors of handwritten digits made by a multitude of classifiers, Pattern Recognition Letters 26 (2005) 369–379.

[21] J. M. Tan, Automatic verification of the outputs of multiple classifiers for unconstrained handwritten numerals, Master's thesis, Concordia University, Montréal, QC, Canada (2004).