**CONTENT**

ABTRACT 1

INTRODUCTION 1-2

ARTIFICIAL NEURAL NETWORK 2-5

VERIOUS STAGES OF RECOGNITION 5-6

HANDWRITTEN DIGIT RECOGNITION 6-8

NETWORK CONFIGURATIONS 9-10

RESULTS 10-11

DIGIT RECOGNIZING DEMONSTRATION 11-12

CONCLUSION 12-13

**Abstract Intro**:

Handwritten digit recognition is one of the important problems in computer vision these days. There is a great interest in this field because of many potential applications, most importantly where large number of documents must be dealed such as post mail sorting, bank cheque analysis, handwritten form processing etc, some of the existing systems include computational intelligence techniques such as artificial neural networks or fuzzy logic, whereas others may just be large lookup tables that contain possible realizations of handwritten digits. So a system should be designed in such a way that it is capable of reading handwritten digits and provide appropriate results. This documentation presents a survey on various neural network approaches to recognize handwritten digits**.**

**A. Introduction:**

Human beings have been given a natural tendency to identify letters, objects, numbers, and voices. However, making a machine do these types of tasks is a very complicated problem. Pattern recognition is a necessary component of artificial intelligence and computer vision. Interest in pattern recognition is growing fast aligned to the prohibitive amount of information that we come across in our day-to-day. Consequently, computerization is very much needed to handle this information explosion [1].

Handwritten digits recognition is an old and important problem of machine learning. The objective is to recognize images of isolated handwritten digits (0-9). More specifically, the problem is equivalent to find a model, which takes image of handwritten digit as input, and output the predicted class label of the image [2]. Furthermore, the ideas and methods to solve this problem would be very useful in various fields of pattern recognition problems where large volumes of real-world data is used [3].

Neural network has been widely used in the field of pattern recognition. This paper presents survey on handwritten digit recognition using various neural network approaches such as Single layer perceptron (SLP) model, Back-propagation (BP) algorithm and Hopfield Neural Network (HNN) model.

***A.1 The approach***

Since the criticism of single-layer perceptrons, mainly for their limitation to building linear separation surfaces [4], more powerful neural network classifiers have been developed. The most popular network is the Multilayer Perceptron (MLP) trained by the backpropagation algorithm [5]. It has been shown in various papers that MLP's with a single hidden layer are universal classifiers, in the sense that they can approximate decision surfaces of arbitrary complexity, provided the number of neurons in the hidden layer is large enough (see for instance [6]). However, there is no simple rule which indicates how many hidden units are required for learning a given task. Moreover, limitations on hardware requirements or computation time may influence the choice of the classifier and favor classifiers with simpler structures and faster training than MLP's.

Classically, the recognition process is divided into preprocessing steps and subsequent classification. Within our approach, the preprocessing operations do not involve learning.

In an earlier paper, we described a procedure, hereinafter termed the STEPNET procedure, whereby any classification problem defined in RN can be decomposed into subproblems which are efficiently solved by a classifier having a single layer of trainable connections [4]. The network is built and trained simultaneously and automatically, without user's intervention. In this paper, we show, using two data bases, the efficiency of the procedure when applied to "real world" problems such as the recognition of handwritten digits. We compare the performances of two classifiers of identical sizes and structures, resulting from the above procedure, operating on the same data bases, with two different data representations: a simple pixel representation and a more elaborate feature representation. While the first data representation results from normalization as the only preprocessing, the second is obtained after feature extraction and normalization.

**B. Artificial Neural Networks:**

Artificial neural networks have been developed since the 1940s, but only the past fifteen years have they been widely applied in a large variety of discipline. ANN is an information processing model of pattern recognition or data classification, through a learning process, which is inspired by the biological nervous systems, such as our brain. It is composed of a large number of highly synaptic interconnected processing elements called neurons or weights, working in coordination to solve specific problems.

Originating from the artificial neuron, which is a simple mathematical model of a biological neuron, many varieties of neural networks exist nowadays. Although some are implemented in hardware, the majority are simulated in software.

***2.1 Learning***

In the learning stage, the network is trained to recognize (parts of) handwritten numerals. This is done by first initializing the network and then repeatedly offering input patterns, determining the winning neuron and adjusting the neurons’ feature vectors (see [7] for a more detailed description).

**Initialization** The self-organizing map consists of *N* ×*N* neurons laid out in a rectangular grid. Each neuron contains (basically *is*) a randomly initialized feature vector with as many elements as the number of inputs, i.e., *M* ×*M*, being the size of the (binary) image that contains (part of) a numeral. The lateral distance between two neurons is the shortest distance between them in the map, usually defined along a straight connecting line or along the gridlines. Depending on the lateral distance between two neurons, they can be either inside or outside each other’s neighborhood. The neighborhoods are important for the adjustment of the feature vectors during learning.

**Finding the Winning Neuron** One by one, training input patterns are offered to the network. For every neuron, the difference (usually Euclidian distance) between its feature vector and the input vector is calculated. The neuron with the smallest difference is then the winning neuron for that particular input example.

**Vector Adjustment** The winning neuron’s feature vector is then adjusted in such a way that the difference with the input example decreases. This is also done for the neurons in the winning neuron’s neighborhood, though the larger the lateral distance from the winning neuron, the smaller the adjustment. Often the neurons directly outside the winning neuron’s neighborhood are adjusted in the opposite direction. The rate of adjustment as a function of the lateral

**Fig.1.** Illustration of a Mexican hat function

distance to the winning neuron has the shape of a Mexican hat, as can be seen in Fig. 1. Over time, this hat is shrinking both horizontally (the neighborhood is getting smaller) and vertically (the adjustments are getting smaller), until the learning finishes.

***2.2 Labeling***

Before the network can classify unknown digits, the neurons need to be labeled. This happens after the learning stage has finished. Several ways of doing this are possible, some of which will be explained below. Once the neurons have been labeled, an unknown handwritten digit can be offered to the network. The label of the winning neuron then serves to classify the unknown digit.

**Majority Voting** Every neuron is assigned a number of counters, one counter for each digit class. Then, from every digit class an equal number of examples is offered to the network. Every time a neuron is the winning neuron, its counter for the corresponding digit class is incremented. After all examples of all classes have been offered, each neuron is labeled according to the class for which its counter was the highest. For example, if a neuron was winning neuron for seven instances of “5”, two instances of “6”, and never for the remaining digit classes, it will be labeled “5”.

Neurons that never were the winning neuron, or for which the above method leads to ambiguities (e.g., as often winner for “5” as for “6”), can be labeled according to the labels of the neighboring neurons using majority voting. This may be done because of the topology preserving property of the self-organizing map: neighboring neurons contain similar information. Alternatively, neurons not labeled in the first round can be labeled “unknown.”

**Minimum Average Distance** Instead of simply incrementing the class counters for winning neurons, more unambiguous labels can be obtained by minimum average distance labeling. In this method, every time an example digit is offered to the network, the corresponding class counters for all neurons are increased by the lateral distance of the neuron to the winning neuron (analogous to the previous method, we use the term “counter” here, even though instead of counting occurrences we now accumulate distances). After all example digits of all classes have been presented, the average distances are calculated by dividing counter for each digit class. Then, from every digit class an equal number of examples is offered to the network. Every time a neuron is the winning neuron, its counter for the corresponding digit class is incremented. After all examples of all classes have been offered, each neuron is labeled according to the class for which its counter was the highest. For example, if a neuron was winning neuron for seven instances of “5”, two instances of “6”, and never for the remaining digit classes, it will be labeled “5”.

Neurons that never were the winning neuron, or for which the above method leads to ambiguities (e.g., as often winner for “5” as for “6”), can be labeled according to the labels of the neighboring neurons using majority voting. This may be done because of the topology preserving property of the self-organizing map: neighboring neurons contain similar information. Alternatively, neurons not labeled in the first round can be labeled “unknown.”

**Minimum Average Distance** Instead of simply incrementing the class counters for winning neurons, more unambiguous labels can be obtained by minimum average distance labeling. In this method, every time an example digit is offered to the network, the corresponding class counters for all neurons are increased by the lateral distance of the neuron to the winning neuron (analogous to the previous method, we use the term “counter” here, even though instead of counting occurrences we now accumulate distances). After all example digits of all classes have been presented, the average distances are calculated by dividing the class counters by the number of example digits for that class. For every neuron, the class with the smallest average lateral distance to the winning neuron forms the label. Neurons that never win but that are close to a winning neuron most of the time, will automatically be labeled accordingly.

**Minimum Average Difference** Yet another labeling method is labeling by minimum average difference. This method is very similar to the labeling by minimum average distance, but instead of accumulating the lateral distances to the winning neurons, we now accumulate the differences between the presented examples and the neuron’s feature vector for each neuron. Again, after all example digits of all classes have been presented, the average differences are calculated by dividing the class counters by the number of example digits for that class. For every neuron, the class with the smallest average difference forms the label.

**\*\*\****In this documentation the section C gives the brief introduction of the various stages of handwritten digit recognition. Section D gives a brief introduction of various artificial neural network techniques used for the classification of handwritten digits. Section E gives the conclusion and future scope of the described work.\*\*\**

**C. Various Stages of Recognition:**

The Handwritten digit recognition system has to go through following step [8]:



*Image acquisition* implies obtaining the image either by scanning documents or by capturing photograph or by directly writing using mouse [9].

*Pre-processing* of the image means applying a number of procedures for the purpose of thresholding, smoothing, filtering, resizing, and normalizing so that further algorithm for final classification can be made simple and more accurate [10].

*Image segmentation* means to divide the image into various segments so that it can be made more meaningful and easy to analyse [11].

*Feature Extraction* is the process to capture appropriate characteristics of the target object. It is mainly related to dimension reduction. The extracted features from an image are applied as an input to the train classifier like neural network [12].

After obtaining feature vector from the input image next step is to decide which classifier to use for classifying the class of handwritten digits. The most traditional classifier to be used is neural network [13].

D. **Handwritten Digit Recognition**

***3.1 Pre-processing***

As digitized numerals may originate from a range of sources, it is important to make sure that all digits are pre-processed so as to eliminate extra problems due to non-class-specific differences in size, shear, line thickness, background and digit colours, resolution, etc. A few techniques to minimize these differences are discussed below.



**Deskewing** To help compensate for different handwriting styles (with different slant), digits that are slanting to the left or to the right are desk wed to put them all upright or to at least make them slant under the same angle. For some examples, see Fig. 2.

The desk wing process works as follows: first, an imaginary line is drawn through the digit so that the total of the horizontal distances from all digit pixels to this line is minimized in a least-squares sense. The imaginary line can be described by *x* = *ay* + *b*, where *x* and *y* are the horizontal and vertical pixel index, respectively, and *a* and *b* are line parameters. Then a side slip is performed: the pixels that form the digit are horizontally shifted to the left or to the right, while keeping their positions relative to the imaginary line constant. This is done in such a way that the imaginary line becomes vertical, with the digit effectively in an upright position. In formula: *xnew* = *xold* −*ay* for every pixel, resulting in the new line *x* = *b*. If the slant is removed by rotation, the recognition problem is likely to become more difficult, due to increased variation.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 10  20  30  40  50 |  | 10  20  30  40  50 |  |

(a) left-leaning (b) side slip (c) rotation

35

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 5  10  15  20  25  30  35 |  | 5  10  15  20  25  30  35 |  |

5 10 15 20 25 30 35 5 1 30 35

(d) right-leaning (e) side slip (f) rotation

**Fig.2.** Two examples of deskewing: (a) and (d): left and right leaning digits, resp.;

(b) and (e): deskewed by side slip; (c) and (f): deskewed by rotation

among numerals from the same category. This may become clear by looking at Figs. 2c and 2 f.

**Thinning** To overcome problems related to differences in the line thickness of the written characters, the digit images are thinned to a thickness of one pixel. This can lead to undesired side-effects, such as spurs, as can be seen from Fig. 3b. To correct for this, one can remove spur lines of up to a certain length from the resulting thinned image (Fig. 3c). A negative side-effect of this pruning process is that end-lines that are actually part of the digit are shortened. The length of these digit parts is usually still long enough for correct recognition of the digit, so that the benefits of the thinning process far outweigh the negative sides. Algorithms exist that try to correct for this ; however, the need for pruning can be avoided by smoothing the handwritten digit image before thinning, as seen in Figs. 3d and 3e. Moreover, this method results in an overall smoother thinned image.

**Resizing** Differences in digit size are due to different image resolutions and handwriting styles. To minimize the effect of these differences, all digits are resampled/rescaled to the same size in pixels, before being fed to the recognizer.

**Smoothing** Some acquisition methods may cause small artifacts on the digit’s edges. These are removed by a smoothing operation. Some pre-processing techniques, including deskewing and resizing, may cause similar artifacts, so smoothing may be useful between several stages of the preprocessing. Effective smoothing can be accomplished by filling tiny gaps in both the foreground .

25

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 5  10  15  20  25 |  | 5  10  15  20  25 |  |

(a) thresholded (b) thinned (c) pruned original (with spurs) (by 5 pixels)25

|  |  |  |
| --- | --- | --- |
|  | 5  10  15  20  25 |  |

(d) smoothed (e) thinned

**Fig.3.** Sequence a-b-c shows the thinning and pruning process; sequence a-d-e shows an alternative: smoothing and thinning without pruning

(a) a gap in the black background (b) not a gap

**Fig.4.** Illustration of how 5 can be significantly more than 3 but 6 not significantly more than 2 (see Sect. 3.1, Smoothing)

background. Filling a gap is nothing more than changing the foreground color of a gap pixel to the background color or vice versa. Tiny gaps in the foreground are background pixels with significantly more neighboring foreground pixels than neighboring background pixels. Tiny gaps in the background are defined in a similar way and can in fact also be seen as tiny foreground protrusions. The term “significantly more” cannot be defined as a simple number, but depends on the layout of the pixel neighborhood. For example, in the layout of Fig. 4a, 5 is significantly more than 3 and the central pixel can be safely smoothed away, whereas in the layout of Fig. 4b, 6 is not significantly more than 2 as the central foreground pixel cannot be smoothed away without breaking the foreground connectivity

.

**4. Network Configurations**

***4.1 Two NNs in Series***

In this setup, one Kohonen map is trained with digit fragments, and another is trained with the ordering information in the trained first map. The idea is that particular combinations of digit fragments are characteristic for a particular digit class; the first network identifies fragments, i.e., not digits, and the second network identifies combinations of fragments, i.e., digits. The second network will need to be labeled in order to be able to classify unknown digits. The first network remains unlabeled. Suppose the digit fragments are *M* ×*M* pixels in size, then the first network (say *N*×*N* neurons), will have *M*×*M* inputs, being the pixel values of the digit fragments, meaning that the vectors stored in the neurons have a length of *M* ×*M*. The second network will then have *N* ×*N* inputs, one for every neuron in the first network. Per digit, for every neuron in the first network is recorded for how many digit fragments that neuron was the winning neuron. These totals are input to the second network.

***4.2 Template Matching Instead of NNs***

Another method for the recognition is to create a template for each digit and to compare unknown digits to these templates to find out which template matches closest. One way of creating such templates is by calculating an average digit from an example set for each digit class.

***4.3 Ten NNs in Parallel***

In this setup, ten networks are trained for one digit class each. Thus, the networks are pre-labeled. They are trained with digit fragments (as before without location information). For the recognition stage, several methods are available, two of which are discussed below. In the special case that the networks consist of only one neuron each and the digit fragments actually contain the whole digit, this method is similar to the template matching method of Sect. 4.2 with the difference lying therein that creating the templates is now realized by training the networks, rather than calculating the average digits directly.

**Minimum Distance Classification** For classifying an unknown digit, (fragments of) that digit are offered to all ten networks, and per network the (Euclidian) distances between the winning neurons and the fragments are totaled over all fragments. Then, the label of the network with the smallest total distance gives the class of the unknown digit. In general, all ten networks will have equal numbers of neurons. If this is not the case, the total Euclidian distance is divided by the number of neurons to find the network with the smallest average distance.

**Label Classification** With label classification, (fragments of) the unknown digit are also offered to all ten networks, but now we are only interested in which network has the winning neuron with the smallest distance to a fragment, measured over all networks. This network is then called the winning network for that fragment. Finally, the label of the most frequently winning network, measured over all fragments, gives the class of the unknown digit.

***4.4 Fragmented Digits vs. Whole Digits***

**Digit Fragments** The idea to use fragments of digits rather than the whole digits, originates from the idea that for a specific digit class, digits as a whole show more variation than individual fragments that are characteristic for that class. When using fragments however, a decision needs to be made whether to include any position information or not. For the purpose of demonstrating the working of a Kohonen self-organizing map, the organization of the fragments is expected to be easier to understand if positional information is left out.

**Whole Digits** When using whole digits as inputs to the networks, what actually happens depends on the training method and the network architecture. With ten small networks, the networks basically store representative examples of the digits during the training stage. These are derived from the example set, although the stored examples are usually not present in the example set themselves, as the network tries to store generalizations of the example digits it receives at its inputs. The networks are pre-labeled and only trained with examples from their respective digit classes.

With one large network, all ten classes of digits are offered to the network during the training stage. The network will then store representative examples of all digits. Depending on the actual size of the network, more complicated numerals or numerals that can appear in several distinctive forms may cause the network to store more examples of that digit class than of another, simpler, class (e.g., consider the class “2”, which can be written in very different ways, and the class “0”, which most people write in much more similar ways, as can also be seen in Fig. 5).

**5. Results**

As the goal was a reasonably well working handwritten digit recognizer for demonstration purposes, and due to very limited time constraints, only a small number of experiments have been conducted to enable comparison between the various configurations. An existing set of handwritten digits with 25 digits per class was used. This set, shown in Fig. 5, was split into a training set of 15(×10) examples and a test set of 10(×10) examples. Previous research ([5] and others) had yielded a maximum 92% correct classification score (percentage correctly recognized digits) on the same test set. Using digit parts, this number now increased to 98% (see Table 1), which was considered sufficient for use in a demonstration. In these experiments, one network was used for each digit class, in total 10 networks. The size of the networks was the same for all 10 networks: 5×5 and 7×7 in two series of experiments. The size of the digits after preprocessing varied between 5×5 and 50×50 pixels and the size of the digit fragments varied between 3×3 and 25×25 pixels. Some results are shown in Table 1. In this table, the best classification results over a range of fragment sizes are shown for a selection of digit sizes, using minimum distance classification (as described in Sect. 4.3).

 

**Fig.5.** Set of handwritten digits used for training (*top*) and testing (*bottom*)

A few experiments were conducted using whole digits rather than fragments. Results of these experiments are shown in Table 2. Per digit class one network was used, and classification was done by minimum distance as described in Sect. 4.3. Shown are the performances (on the test set) per digit size for two network sizes. The results are worse than in the experiments with fragmented digits, but strong conclusions cannot be drawn due to the limited number of experiments.

**6. Digit Recognizing Demonstration**

A continuous presentation has been installed at the Da Vinci Techno Activity Center in Enschede, the Netherlands, where visiting children and adults can learn more about technology. The presentation consists of a slide show explaining the whole Kohonen system of learning and recognizing, and a writing pad with display where visitors can write a digit and see the NN outcome. This NN is the same one that scored best in the experiments: 98% correct recognition rate, see Table 1. In the slide show, a set of unclassified digit fragments is shown

**Table 1.** Results of experiments with fragmented digits

|  |  |
| --- | --- |
| network size | digit size [ pixels ] |
| 5×5 10×10 15×15 20×20 25×25 |
| 5×5  7 7 | 73% 93% 98% 94% 93 %  73% 90% 92% 86% 82 % |

**Table 2.** Results of experiments with whole digits

|  |  |
| --- | --- |
| network size | digit size [ pixels ] |
| 5×5 10×10 15×15 20×20 25×25 |
| 5×5  7 7 | 63% 82% 91% 92% 93 %  70% 83% 92% (not used) |

to give people also a chance to determine what digit the fragments come from. The whole presentation was part of “NeuroFuzzyRoute in the Euregio,” an exposition in the framework of the world exposition EXPO2000 in Hannover.

Unfortunately, the digits written by visitors of the techno activity center have not been recorded, nor is anything known about the recognition rate. However, visitors have reacted enthusiastically to the hands-on demonstration.

**7. Conclusions**

We have shown that our network, resulting from the MLP building and training procedure and using an appropriate data representation, leads to very satisfactory recognition rates on two moderately sized data bases of handwritten digits. A priori knowledge about the classification task is used to design explicitly the preprocessing steps; the classifier itself is very simple, featuring 45 binary neurons and a few logic gates. The performances of our classifier on "real world" data bases appear to be at the standard level of present-day recognizers, i.e. roughly a 10% rejection rate for a 1% error rate in the recognition of handwritten digits without constraints on writing style. Yet, in comparison to other networks, e.g. Multilayer Perceptrons, the structure of the discussed network is simpler, and it is generated automatically. In addition, the STEPNET procedure gives some insight into the difficulty of the classification problem: in the case of handwritten digits, it indicated that, surprisingly enough, the sets of examples were pairwise linearly separable. Both training times and classification times compare favorably with respect to Multilayer Perceptrons, and the simple structure of the network as well as the use of binary neurons facilitate greatly hardware implementation. An integrated circuit, performing the classification of a digit in 130 碌s, has been designed and fabricated; its speed makes it a good candidate for performing the classification task necessary for the automatic segmentation of zip codes.This documentaation shows that a Kohonen self-organizing map can be used to recognize digits when given those digits in fragments without positional information. The Kohonen self-organizing map and the handwritten digit recognition have also proven to be a good neural network architecture and application for the purpose of introducing and demonstrating neural networks to the general public.

.