

Optical Character Recognition: Neural Network Analysis of Hand-Printed Characters

Adnan Amin¹ and Sameer Singh²

¹ School of Computer Science and Engineering
University of New South Wales, Sydney 2052, Australia
e-mail: amin@cse.unsw.edu.au

² School of Computing, University of Plymouth
Plymouth PL4 8AA, United Kingdom
e-mail: slsingh@plym.ac.uk

Abstract: The main objective of this paper is to introduce a novel method of feature extraction for character data and develop a neural network system for recognising different Latin characters. In this paper we describe feature extraction, neural network development for character recognition and perform further neural network analysis on noisy image segments to explain the qualitative aspects of handwriting.

1. Introduction

The recognition of hand-written characters using artificial techniques including neural networks is an important area of research for realising commercial advances in OCR technology. In the past neural networks have been used for character recognition for classifying written input from different sources. In this paper we describe a novel method of feature extraction for recognising Latin characters. The character image is segmented for extracting primitives in each segment. The extracted primitives are used for training a three layered neural network for recognising unseen test samples. The results obtained using cross-validation show very high recognition rates using this approach. Neural networks are further used to analyse the effect of noise on recognition rates when individual image segments are affected by gaussian noise. This analysis yields important information on the writing behaviour of individuals and off-line recognition of characters in noisy environments. We first describe feature extraction followed by the neural network analysis.

2. Feature Extraction

First, sample characters are digitized. The digitization algorithm is similar to that which appears in [1]. A 300 dpi scanner is used to digitize the image. Next, the digitized image is cleaned and thinned by using parallel algorithm [2]. Pre-processing is followed by feature extraction. The heart of a feature extraction process for all character recognition systems is determining the primitives features. In our system, we use two main primitives, straight line and curve.

2.1 Binary tree construction

An algorithm implementing a 3x3 window is used to trace along the path of the skeleton, recording the structural information of the trace path. A path is described as a

tracing between junction or end point, where an end point has a single neighbour and a junction point has two neighbours as shown in Figure 1.

This path is stored in a node of the binary tree: where a choice of path to trace exists, a left and right node are formed beneath the current one and their respective paths traced out. A priority system is used which favours certain directions over others (without this, the window would trace the skeleton in random direction).

This path is stored in a node of the binary tree: where a choice of path to trace exists, a left and right node are formed beneath the current one and their respective paths traced out. A priority system is used which favours certain directions over others (without this, the window would trace the skeleton in random direction).

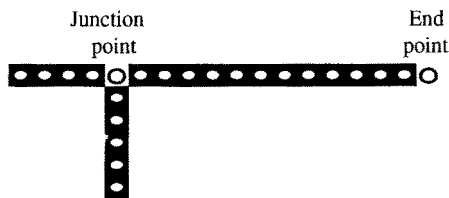


Fig. 1: Tracing path.

This path is stored in a node of the binary tree: where a choice of path to trace exists, a left and right node are formed beneath the current one and their respective paths traced out. A priority system is used which favours certain directions over others (without this, the window would trace the skeleton in random direction). The starting point for tracing the skeleton is based on several criteria. The image is divided into three horizontal regions and the top and bottom region are searched for end points or junction points. This ensures that the starting point does not split a path into two subpaths. If no such points are found, as with the letter "O", the left most pixel of the image is used as starting point.

2.3 Structural information

The structural information for each path traced is saved as follows:

1. *Freeman Code* [3] chain: an 8-directional code describing the tracing of the path.
2. *Positional*: co-ordinates describing the start and ending points of the path used to determine positional relationship between loops and between loops and touching path (e.g. "b" = left touching path, "d" = right touching path).
3. *Loop*: pointer indicating paths joining previously explored section (e.g. "e", "6")

The completed tracing results in the segmentation of the character into paths or strokes which will be formed into primitives.

2.4 Primitives

The structure information in the binary tree allows the formation of pattern primitives, or sub-pattern, which are used to describe the original image. There are two main primitives described in this system: straight lines and curves. A path may be described by a single primitive or by multiple primitives. The structure information in the tree is converted to these primitives using the following definition.

Breakpoint (Separator): divides a path into sub-paths more easily described by primitives. A breakpoint has at least one of two possible conditions:

- inflection point: a change in curvature, a positive (clockwise) curve followed by a negative (anti-clockwise) curve, or vice versa.
- cusp point: a sharp change in direction, two segments form an acute angle ≤ 90 .

Straight line: has its usual geometric definition as two points in sequence within a path. A point in a Freeman chain can be defined as a change in the Freeman code. Lines can be distinguished from curves in two ways: the length of a line segment is significant in comparison to the length of the path, or the path contains only two points.

Curve: these are formed by at least three segments of nearly equal length (usually small in relation to the length of the path) with no breakpoint. Two types of curves: Open and Closed.

- Open: there are four open curves useful in describing Latin characters. They face the four main points of the compass, eg. U, S, C.
- Closed (Loop): e.g. a, R, D, 8.

The primitives used for this study are shown in Figure 2.










Line				
Open curve				
Closed curve				

Fig. 2: Primitive features used in this system

3. Experimental results

Hand-printed character data is highly non-linear in nature as a result of varying styles of writing by different participants. For our analysis, we therefore need to use an efficient non-linear classifier for recognising character data. In this paper we have used a three layered neural network for this purpose. Neural networks are selected because of their well known generalisation capabilities and ability to characterise non-linear data. We describe three phases of analysis for using a neural network for character recognition: data pre-processing and input data selection, neural network architecture and algorithm selection, and recognition results obtained with a cross-validation study and noisy character data.

3.1 Data pre-processing and input selection

The initial data sampled for the character recognition exercise consisted of a total of 1115 patterns for a total of 52 classes with roughly 21 patterns in each class. As described in earlier sections, a moving window of fixed size traces the presence or the absence of the nine primitives in a sequential manner. The initial position of the

window is dependent on the free points found in a character or if the character has an unbroken loop, it starts from the top left corner. For each window position (a total of twelve), the presence of already discussed nine primitives is traced in a sequential manner. If a primitive is found, then its code is recorded in a stack. For no primitive found, an “empty” code is recorded. A data pattern consists of the sequence of codes available in the stack at the end of the analysis. For different writers, a particular written alphabet should have a similar code sequence for different samples (writers). It is assumed that the difference between different writers writing the same character is much smaller in comparison with different writers writing different characters. Our initial analysis shows that for five of the twelve measurements made in different window positions, in most cases there were either no primitives present in those windows, or the primitives found did not change with different characters significantly to be considered important for the classification process. For this reason, we finally settle on using measurements recorded in a total of seven window positions which record the presence of already described nine primitives for the purposes of classification.

Each class represents a particular alphabet with the first twenty six classes representing lower case letters (a-z) and the next twenty six classes the same alphabets in upper case (A-Z). A total of nine primitives are extracted for character analysis as described in the previous section for a total of seven image windows (Figure 2). For each input vector $P = \{x_1, x_2, \dots, x_7\}$, its target output was represented as the vector $\{t_1, t_2, \dots, t_{52}\}$. Here x_i ($1 \leq i \leq 7$) represents the i th window measurement and t_j ($1 \leq j \leq 52$) is the target output for supervised learning, i.e. if $t_1 = 1$ and all other t_j are 0, then the input pattern is “a”. Similarly, if $t_2 = 1$ and all other t_j are 0, then the input pattern represents a “b”. In this order, if the target output $t_j = 1$ in position j , then it represents a hand-written character of class j in sequence a...zA...Z. The data finally presented to the neural network for training consisted of all 1115 input patterns with their respective target patterns. For all of our results, the data has been interleaved before presentation, i.e. class1 pattern is followed by class2 pattern, followed by class3 pattern and so on. This ensures better training since it allows the weights to adjust for the given problem in a well balanced manner. The Stuttgart Neural Network Simulator (SNNS) used for this study also offers the facility to “Shuffle” or randomly mix input patterns before presenting them to the neural network. This facility was also used during network training.

3.2 Neural network development

In the selection of an appropriate architecture and for further analysis, we first determine the least complexity model with minimal training error as recommended by Weiss and Kulikowski [4]. For this purpose, we train a number of neural network architectures with the same algorithm but with varying number of hidden nodes and choose a model which produces the minimal training error with the character data. We find that using the recommended procedure, a $7 \times 120 \times 52$ architecture is adequate for our purposes. For this architecture, there are seven nodes in the input layer, one hundred and twenty hidden nodes and fifty two output nodes. For training the neural network, backpropagation with momentum training method is followed. This method was selected because of its simplicity and because it has been previously used on a

number of pattern recognition problems. The method works on the principle of gradient descent and has been described in its basic form in detail by Rumelhart et al. [5]. The algorithm uses two parameters which are experimentally set, the learning rate η and momentum μ . These parameters allow the algorithm to converge more easily if they are properly set by the experimenter. Bishop[6] describes backpropagation with these two techniques in detail. In order to arrive at a reasonable values of these parameters for training we have taken into account the guidelines set by Vogl et al. [7]. We finally settle on a learning rate $\eta = .77$ and a momentum term $\mu = .9$. This algorithm is available with the Stuttgart Neural Network Simulator (SNNS) which has been used for our experimentation throughout.

3.3 Pattern recognition with cross-validation

The neural network analysis of data is performed in two separate phases: training, when the network learns by example in an iterative manner, and testing, which then presents unseen data to be classified. In order to measure the performance of our neural network system on the character recognition problem, we use the cross-validation procedure.

Fu [8] describes the cross-validation process as: “*K*-fold cross-validation (Stone[9]) repeats *K* times for a sample set randomly divided into *k* disjoint subsets, each time leaving one out for testing and the others for training”. The value of $K = 10$ is usually recommended [4]. Cross-validation requires that the original data set is split in *k* disjoint sets. At any one time, 90% of the data is used for training and the performance is tested on the remaining 10%. Each training process is called a ‘fold’. At the end of 10 folds, all data has been tested. In every fold therefore the training and test patterns are different. The overall performance of the system may be measured using two different parameters: the average recognition data of training data in percentage (av. R_α), and the average recognition rate of the test data in percentage (av. R_β). As expected, the latter is smaller in practice but is very important since it represents the true performance of the neural network.

Table 1 shows the recognition performance using ten-fold cross-validation. Here the recognition rate of the training and test data at the end of fold K ($1 \leq K \leq 10$) training is shown in separate rows of the table. The recognition rate in percentage represents the ratio of the total number of correctly classified patterns to the total patterns tested during a test phase. We follow rigid guidelines for specifying what is a correct classification. For a test pattern whose target is $\{t_1, t_2, \dots, t_{52}\}$ and the actual output is $\{T_1, T_2, \dots, T_{52}\}$, the correctly classified pattern must satisfy the condition $T_j - t_j < 0.2$ for all j ($1 \leq j \leq 52$). If this condition is violated even once in a pattern, then it is misclassified. Similar stringent guidelines are followed for training. The training process for the network is stopped only when the sum of squared error falls below .0001.

It is important to note here that the system performs extremely well with recognition rates ranging between 84% and 88% on different folds and the overall recognition is 85.8%. This is a very good performance taking into account the fact that we have a limited number of samples in each class and that we have not used any noise filtering techniques. A linear discriminant analysis yields a recognition rate of 56% at

best. The recognition on the training data is also extremely high, 99.8% which represents very good training.

In Table 1, the results have been produced keeping the experimenter bias to a minimum when developing a neural network for analysis and the feature extraction stage. These set of results however do not tell us about the quality of our feature extraction in terms of their resistance to noise. In other words, we need to quantify how well the system will perform in the presence of noise. For this purpose we generate gaussian noise with a fixed distribution (mean = 0, sd. = 1) and use this to contaminate our character recognition data. Recognition rates are then recorded for varying noise amplitude. For further experimentation we do not follow cross-validation since our aim is not to investigate the true generalisation error, rather it is to quantify the degradation in performance with pre-defined step-wise increases in noise. For this purpose, data in fold 2 in Table 1 is selected (marked with an asterisk). We train with 90% of the data in the training set and test with 10% of the data in the test set, when injected with additive non-cumulative noise of varying amplitude. The noise data is generated using a C++ function library. The noise vector N is a series of randomly generated numbers which is transformed within the $[-1, +1]$ range. A total of ten trials are conducted, each time varying the maximum offset allowed. The maximum noise offset δ represents the maximum noise possible for a single pattern. The actual noise value for a particular pattern with the $[-1, +1]$ range is multiplied by this maximum offset before being added to the character data. The average noise \bar{N} represents the ratio of the total noise present in the data and the number of patterns. This value for a particular trial is always much below the noise offset δ for that trial. Since noise is random, the average noise \bar{N}_α for training data is different to the test data \bar{N}_β . For different trials, we use different noise series but with the same noise distribution. During each trial, the neural network is trained and tested with noisy character data. Table 2 shows the recognition results obtained using the above procedure.

Fold	Recognition Rate % Training (R_α)	Recognition Rate % Testing (R_β)
1	99.9	85.0
2*	99.9	87.0
3	99.7	87.0
4	99.8	84.0
5	99.8	86.0
6	99.9	86.0
7	99.9	85.0
8	99.9	85.0
9	99.9	88.0
10	99.8	85.0
Average	99.8	85.8

Table 1. Neural Network recognition rate performance using ten fold cross-validation. Recognition rates on the training and test sets.

In Table 2, the average noise per pattern added to both the training and test set is shown with the recognition rates obtained on both the training and test set when the neural network learning was finished. As expected, in every successive trial, the amount of noise added to the system increases. The training recognition rates fall and then start to rise: this phenomenon has been noted in other studies when the presence of noise actually helps the neural network for training purposes [10, 11]. Following this trend, the test performance also degrades with increasing noise for most cases. Some important points of observation may be stated as follows:

- The degradation in performance is graceful and predictable. The correlation r between the amount of noise and the drop in recognition rate is high, $r_{\text{train}} = -.85$, and $r_{\text{test}} = -.86$.
- The recognition rates are high for most trials except when the noise increases considerably.
- The degradation in training and test recognition rates are highly correlated $r = .97$ but in most cases the degradation in performance is not directly proportional.

Trial	Noise	Av. α Noise \bar{N}_α	Recognition Rate % Training (R_α)	Av. β Noise \bar{N}_β	Recognition Rate % Testing (R_β)
1	.1	.02	99.9	.03	85.0
2	.2	.06	96.7	.07	81.0
3	.3	.07	93.0	.08	74.0
4	.4	.10	90.5	.12	75.0
5	.5	.128	87.6	.13	71.0
6	1.0	.28	73.7	.37	58.0
7	1.5	.40	76.2	.45	56.0
8	2.0	.51	66.8	.61	51.0
9	2.5	.59	68.2	.90	48.0
10	3.0	.70	77.7	1.23	52.0

Table 2. Neural Network recognition rate performance with noisy hand-written character data. Results are shown for gaussian additive noise added to *both the training and the test set*.

4. Conclusion

The technique adopted in this paper for recognizing hand-printed Latin characters using a neural network combined with a structural approach. The method is based on structural primitives such as curves, straight lines and loops, in a manner similar to that in which human beings describe characters geometrically. Moreover, we have used neural networks for recognizing different hand-printed Latin characters using a cross-validation study. In our study with ten fold cross-validation study, for most folds we obtain recognition rates as high as 88% on unseen test data and the overall average recognition rate for the test data is nearly 86%. We have also used neural networks for further analysis when the character data is contaminated by noise of varying amplitude. The results show that the degradation in performance is graceful and predictable.

References

1. A. Amin, A. Rajithan and P. Compton, Recognition of handwritten characters using induct machine learning, 6th International Workshop SSPR'96, Germany, Springer, 189-197, 1996.
2. B. K. Jang and R. T. Chin, One-pass parallel thinning: analysis, properties, and quantitative evaluation, *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-14**, 1129-1140, 1992.
3. H. Freeman, On the encoding of arbitrary geometric configurations, *IEEE Trans. Electronic Computers* **EC-10**, 260-268, 1968.
4. S. M. Weiss, and C. A. Kulikowski, Computer systems that learn, Morgan Kaufmann, San Mateo, CA, 1991.
5. D. E. Rumelhart, G. E. Hinton. and R. J. Williams, Learning internal representations by error backpropagation, In D. E. Rumelhart and J. L. McClelland (eds) *Parallel distributed processing*, vol 1, 318-362, MIT Press, 1986.
6. C. M. Bishop, *Neural networks for pattern recognition*, Clarendon Press, Oxford, 1995.
7. T. P. Vogl, J. K. Mangis, A. K. Rigler, W. T. Zink, and D. L. Alkon, Accelerating the convergence of the back-propagation method, *Biological Cybernetics*, **59**, 257-263, 1988.
8. L. Fu, *Neural Networks in Computer Intelligence*, McGraw-Hill. Singapore, 331-348, 1994.
9. M. Stone, Cross-Validatory Choice and Assessment of Statistical Predictions, *Journal of the Royal Statistical Society.* **36** (1), 111-147, 1974.
10. M. Burton, and G. J. Mpitsos, Event-dependent control of noise enhances learning in neural networks, *Neural Networks*, vol. **5**, 627-637, 1992.
11. K. Jim, B. Horne and C. L. Giles, Effects of noise on convergence and generalisation in recurrent networks, *Neural Information Processing Systems* **7**, G. Tesauro, D. Touretzky and T. Leen (eds.), MIT Press, p. 649, 1995.