Next Generation Optical Character Recognition using the Polynomial Method

Sam Danziger[1]
Computer Science Department
Electrical Engineering Department
Rochester Institute of Technology

August 16, 2002

[1]Sam_Danziger@ieee.org

Abstract

This project extends on the work done Drs. Anderson and Gaborski' in Optical Character Recognition (OCR) of handwritten digits done back in the 1990s. Their work involved using the polynomial discriminant method, iterative training, and genetic algorithms to optimize the feature selection.

This project involves several variations in technique over previous work. The first is the use of the matrix manipulation language Matlab^{TM} rather than 'C' This allowed the faster training of the matrices (due to highly optimized matrix manipulation functions) and rapid debugging due to a built in debugger and other high level language advantages.

This project also utilized a less random form of feature selection than the "knight moves" or other feature selection techniques used previously. This technique involves compressing the digits from $30X20$ pixels into $8X5$ and $6X4$ "fat pixels" and then taking all possible combinations of pixels between the two compressed digits (resulting in 960 features)

This project also explored the creation of a network of classifiers trained to differentiate between between all possible pairs of digits (0,1), (7,8), et. al. This resulted in ${}_{10}C_2 = 45$ classifiers used in tandem to recognize the hand written digits.

The Polynomial Classifiers initially created were very similar to those produced previously with one major exception: When the 960 features were reduced to 300 features using the genetic algorithm, the weight matrix proved to be surprisingly skilled at identifying previously unseen digits. For example, when classifying digits not used to train the matrix, the 960 features produced a 43% accuracy while the 300 features produced a 75% accuracy.

By using a larger training set, and training for more iterations, this accuracy was improved to nearly 80% correctly classified digits. The ${}_{10}C_2 = 45$ network classifier performed best of all classifying up to 83% of unseen digits correctly, opening up a whole new area for further research.

0.1 Theory

0.1.1 Feature Selection

Optical Character Recognition (OCR) begins by somehow scanning and isolating individual characters so that they are represented in some sort of digital format (such as bitmap or JPEG). For this experiment, 98,558 handwritten characters were encoded as 30 pixel by 20 pixel binary images (called digits) (See Figure 1).

Given these 600 pixels, each with 2 possible states, there are $2^{600} \approx 4.15 \times 10^{180}$ which is too many states to reasonably compare against to classify digits. One method of OCR involves choosing several features (sets of pixels) from the 600 pixels. These pixels are then used with a weight matrix or neural network (See Subsection 0.1.2).

If pairs of pixels are selected to create a feature, there are $600 \times 599 = 359400$ possible features. To narrow this down, the number of pixels was reduced by compressing the digit. This was accomplished by sectioning the $30X20$ pixels off into $4X4$ and $5X5$ "Fat Pixels". To make the Fat Pixels even encompass the image, it was necessary to pad background pixels onto the edges of the digit (See Figure 2).

These fat pixels were then compressed by summing up the value of all the sub-pixels and saving the result as $8X5$ (40 Pixel) and $6X4$ (24 Pixel) compressed digits (See Figure 3). If features are created by combining (multiplying) all possible pairs of pixels between these two compressed digits (as they were), there are $40 \times 24 = 960$ features, a much more manageable number.

0.1.2 Polynomial Network

A collection of D input digits was evaluated at these 960 (N) features to create the input set X which is used as the inputs to a Neural Network like weight matrix W. Each X is associated with a 10×1 output vector with a 1 in one position to indicate the correct digit. The collection of output vectors is called Z.

$$W[10 \times N] * X[N \times D] = Z[10 \times D]$$

With this a general purpose OCR program can be created, an optimal 'W' will be required that can correctly recognize the digits in 'X' and new unknown digits.

Due to Matlab^{tm} implementation details (see next section), it was useful to rewrite the equation by using the transpose of the X and Z matrices:

$$X^T[D \times N] * W[N \times 10] = Z^T[D \times 10]$$

One imperfect solution is to calculate W by multiplying both sides of the equation by the pseudo-inverse of X, $X^{\#}$.

$$W = (X^T)^{\#} * Z$$

To actually perform the pseudo inverse, it is necessary to get a square matrix on the right side of the equation, so that its inverse may be taken.

$$X * X^T * W = X * Z$$

Let $A[N \times N] = X[N \times D] * X^T[D \times N]$ and $B[N \times 10] = X[N \times D] * Z^T[D \times 10]$.

$$B * W = A$$

Multiply both sides by the inverse of B (B') to calculate W.

$$W = B' * A$$

Unfortunately, the weight matrix produces an imperfect result. Meaning, when you perform the reverse multiplication

$$Z_2 = (X^T * W)^T$$

the new answer $Z_2 \neq Z$. How close Z_2 is to Z depends on how big N compared to D is (as more digits are added with the same number of features, the weight matrix become less accurate), and how dissimilar the digits being compared are. For example, the pair $(0, 1)$ are very dissimilar, and produce a pretty good weight matrix. On the other hand $(6, 8)$ are very similar, and produce a comparatively poor weight matrix.

Regardless, the weight matrix can be trained to produce better results.

0.1.3 Iterative Learning

The weight matrix is trained by an iterative process. Each iteration, Z_2 is compared to Z to see which digits were incorrectly classified. The incorrect

digits are placed in a new X (called X_T) and a new Z (called Z_T). From these, a A_T and B_T are created.

$$A_T = X * Z_T$$

$$B_T = X * X_T$$

The original A and B are then recalculated by adding A_T and B_T and then a new W is produced (0.1.2.

$$A = A + A_T$$

$$B = B + B_T$$

This new W is used to produces a new Z_2 until an acceptable W is produced.

0.1.4 Genetic Algorithms

Once an acceptable W is produced, it is desirable to reduce the number of features to a number smaller than 960. For the purposes of this experiment, a genetic algorithm was used to reduce the number of features down to 300.

The genetic algorithm worked as follows: A population of 20 or 30 individuals with 300 elements was created. Each of those 300 elements was a unique index between 1 and 960 that indicated which of the 960 features should be used to recalculate the weight matrix.

Essentially, this means taking a subset of the X matrix. This can be done by taking subsets of the A and B matrixes, and then recalculating W. (See Figure 4 and Figure 5 for more details).

These individuals are assigned a fitness value based on how close the calculated Z_2 matrix is to the original Z. In effect, the fitness value is a percentage correct. Parts of the individual strings with higher fitness values (parents) are combined (in a technique called crossover) to produce (eventually) more fit children until 300 features are found that have a fitness value nearly as high as the 960 features.

The crossover technique had to be such that it preserved 300 unique features in each individual (see the following Matlab^{TM} for more details) , even though two parents are likely to have many features in common.

0.1.5 45 Classifier

The above techniques were capable of producing a reasonable classifier in a reasonably short period of time (less than a day to train 14388 characters) and produce up to 80% accurate results classifying a set of digits that was not used for training.

However, a technique was desired that could train with the entire 95885 characters and produce better results. Towards this end, there was an attempt to create a classifier that would differentiate between all possible pairs of digits, rather than all 10 digits at once.

This classifier was called the ${}_{10}C_2$ or 45 classifier after the 45 different weight matrices required. The "guesses" from these 45 matrices were tallied and whichever number received the most guesses (9 guesses) was the value indicated by the classifier.

0.2 Matlab

The language Matlab^{TM} was chosen to implement this research into the Polynomial Classifier. The Polynomial Classifier OCR method is essentially a large matrix manipulation problem, and Matlab^{TM} was designed for matrix operations. The biggest advantage is that it includes the highly optimized built in \ operator. The Matlab^{TM} code,

```
W = B\A
```

is equivalent to

$$W = B' * A$$

but by using back-substitution and very likely machine code optimizations it is able to perform this operation very quickly. It should be noted that Matlab^{TM} requires B to be a square matrix to use the \ operator correctly. It takes .07(s) for a $300X300$ B matrix, much faster than even compiled 'C' code.

This section contains an outline of the important Matlab^{TM} functions (.m files) and a brief description of their purpose and any important features they contain.

0.2.1 ocr_main1.m

This file contains the main code for running the code that runs the iterative training for the polynomial matrix. It loads the digits, creates the features, evaluates the fitness, and tests the resulting weight matrix against untested values. It does most of this by calling subfunctions.

0.2.2 ocr_compress.m

This function takes the raw $20X30$ digits stored in the text file, and compresses and saves the digits compressed into $10X7$, $8X5$, $6X4$ and *raw* (uncompressed) .mat files. It creates one file for each digit (0-9) and each type of compression. Once this was run once, it was not necessary to run it again.

0.2.3 ocr_getDigits.m

This function loads the digits from the files created in ocr_compress for use in the training and evaluating of the classifier. It loads two digits matrices and two Z matrices, one pair for training, the other for testing.

It is called twice, once to load the $8X5$ digits and a second time to get the $6X4$ digits.

0.2.4 ocr_genX.m

This function takes two sets of digits ($8X5$ and $6X4$) and produces all (960) feature pairs. In short, it creates the X matrix.

0.2.5 ocr_calcAB.m

This function is the single most intensive function both in processor time and memory requirements. It provides the large matrix operations necessary to compute A and B. During the course of these experiments, memory limitations with Matlab^{TM} and ultimately the x86 Windows architecture were exposed. This function is where the program would crash if it ran out of memory.

0.2.6 ocr_calcFV.m

This function is the most called. As per it's name, given an A, B, X, and Z matrix, it calculates W then creates a Z_2 and evaluates it fitness, a confidence value, and a list of which digits are in error.

0.2.7 ocr_trainMatrix.m

This function adds the incorrectly classified digits back into the A and B matrices.

0.2.8 ocr_GA.m

This function is the main function for reducing the 960 features to 300 features using a genetic algorithm. It takes the A and B matrices calculated in ocr_main1.m.

0.2.9 ocr_GA_genpop.m

This function generates the initial population for the genetic algorithm.

0.2.10 ocr_GA_findComp

To get the subset of A and B Matlab^{TM} uses code similar to

```
Anew(idxs,:)=[]
```

This removes all rows specified in 'idxs' from the matrix *Anew*. Since ocr_GA_genpop generated set of 300 features to keep, this function was necessary to generate the 660 features to throw away.

If an individual contained some number of unique features other than 300, the individual would specify 300 features (some of them repeats), but this function would cause A and B to be reduced to less than 300 features.

0.2.11 ocr_GA_getSubset.m

This function uses the complement generated in ocr_GA_findComp to get the subsets of A and B

0.2.12 ocr_GA_tourney.m

This function chooses 4 random individuals and picks the two most fit (parents) and the two least fit (future children).

0.2.13 ocr_GA_cross.m

This function crosses the two parents found in ocr_GA_tourney and returns two children. The crossover method preserves exactly 300 unique features.

It finds all unique feature pairs in the combination of both parents, which will be between 300 and 600 features pairs. It creates a random list containing all of these features. It then appends another copy of the list onto the end of the list, creating a list that is 600 to 1200 characters. The first child is the first 300 characters, and the second child is the second 300 characters.

0.2.14 ocr_45_run.m

This function calls ocr_main1 and ocr_GA to train all 45 weight matrices for the ${}_{10}C_2$ classifier. It produces a lot of additional information that can be used for profiling and the like. Unfortunately, this means that the raw results of running this function are about 4 GB. It also takes about 3 days to run using all 95885 digits given a 666Mhz processor and 512MB of RAM.

0.2.15 ocr_45_extract2.m

This function extracts the weight matrices from the results of ocr_45_run and and saves them in a convenient format about 1MB in size.

0.2.16 ocr_use45_script.m

This uses the ${}_{10}C_2$ classifier to see how well it works.

0.3 Results

Some notes on how to read Figures 6 - 15. These figures show the accuracy of a trained weight matrix in recognizing it's training set, and where appropriate a series of untrained digits.

0.3.1 OCR Weight Matrices

Figures 6 - 11 are a progression of weight matrixes. These matrixes have been trained with the 960 features discussed previously using the iterative training method. These 960 features were then reduced to 300 features using a genetic algorithm.

The title of each graph contains text that resembles

```
Polynet trained to 2882 characters (Iter: PN = 175 GA = 100)
```

This means that this weight matrix was trained using 2882 digits, and the the Iterative Polynomial training method was run for up to 175 iterations and the Genetic Algorithm was run for up to 100 iterations.

The X-axis has a value called confidence. The confidence is the maximum value in each column (each column being the collection of features for a given digit) of the Z_2 matrix divided by the sum of all values in that matrix. In other words, it is how certain the classifier is that it correctly classified a digit.

The Y-axis has a percentage correct. This is the cumulative percentage of correct guesses at each confidence level.

The legend contains somewhat cryptic text such as

```
PN-T, %cor = 99.91, ct = 0.4577, %act = 94.69
```

- PN: Polynomial Network: Means that this line used all 960 features for the classification. It could also be GA (Genetic Algorithm) which means that 300 features were used for classification.
- T: T means this test uses digits the weight matrix was trained with. A U would mean digits the weight matrix was not trained with were used.
- %cor: This is the percentage of digits correctly classified in this test case.
- ct: This value is the Confidence Threshold. Any digit classified with a confidence value higher than ct was always correct.
- %act: This is the percentage of the digits in this test case that had confidence values above the confidence threshold.

0.3.2 45 Classifier

Figures 12-14 show data from the ${}_{10}C_2$ classifier. The first two figures show individual weight matrices from the classifier network evaluated in a manner similar to that for the standard weight matrices. The only real difference is that since all 95885 available characters were used to train Figure 13, there were no untrained test character available.

For Figure 14 and Figure 15 the meaning of confidence value is slightly different. Since the result is the collection of the results of 45 different matrices, the confidence is the sum confidence of all 45 matrices, normalized.

0.4 Analysis

Each of the figures shown in results contains information within itself and as a part of the whole. For that reason, it is helpful to analyze each figure in it's own section.

0.4.1 Proof of Functionality

Figure 6: This weight matrix established that using the techniques outlined in the theory section would successfully produce a weight matrix. Due to the small number of characters, it was easy (and quick) to train the 960 feature matrix up to 99.9% accuracy and to select some decent 300 features.

It should be noted that given this training, when subjected to a large set of unseen characters, the classifier did pretty poorly, only correctly classifying 43% of the digits. However, when the same set of unseen digits were evaluated using the reduced 300 feature set, the accuracy improved by over 30% to 74.53% correctly classified.

While the confidence seemed like an interesting idea when testing the classifier versus it's training set, when the weight matrix was used to classify unseen digits, only a very small % of the digits were above the confidence. This makes using the confidence as an assurance that the classifier was correct not a very useful idea.

0.4.2 More Training

Figure: 7: This weight matrix was trained with more digits and the iterative training was allowed to run for more iterations. This produced a large

improvement in the classification of untrained digits using 960 features, but the reduced 300 feature continued to show much better performance.

0.4.3 Asymptotic Behavior

Figure 8 and Figure 9: This pair of matrices use a yet larger training set (4799 digits) and only vary the number of iterations the genetic algorithm is allowed to run for. The increased number of digits improves the performance of the 960 feature classifier to the point where it is only about 15% worse than the 300 feature classifier. However, the 300 features is starting to show signs of overtraining. Increasing the number of iterations from 200 to 1000 produces a slight increase in the fitness value versus trained digits, but a slight decrease in fitness versus untrained digits.

0.4.4 A More Intensive Run

Figure 10: This weight matrix was given a large character set an allowed to run for many iterations. The large set of training digits seemed to greatly improve the ability of the 960 feature set to classify untrained digits, bringing it up to 71.5%. However, the 300 feature set continued to outperform the 960 features versus untrained digits classifying nearly 80% correctly.

0.4.5 A Most Intensive Run

The graph here is just a place holder. The real graph will be added tomorrow.

0.4.6 Building the 45 Classifier

Figure 12 and Figure 13: The ${}_{10}C_2$ classifier requires 45 well trained weight matrices. To construct these, principals discovered in previous experiments were used:

1. The 300 feature classifier worked much better in classifying unseen digits. Therefor, it was the better method.

2. It is not useful to run the GA for many iterations to produce the 300 features. Therefor the number of GA iterations was limited to 1000.

3. It was however useful to train the classifier with as many digits as possible. Toward this end two ${}_{10}C_2$ classifiers were built. One was trained with all possible digits (95885 in all), the other was trained with only 25% of the available digits so that there would be plenty of untrained digits to test it against.

These two figures demonstrate the behavior of the binary classifier that decides between '1' and '7' (chosen as an example). These classifiers performed very well.

0.4.7 The 45 Classifier

Figure 14 and Figure 15: These figures demonstrate the ${}_{10}C_2$ classifier in action. When trained with all 95885 digits, the classifier did okay in recognizing the training set, getting about 93% correct. This is slightly worse that the 300 feature classifier usually did classifying the training set.

It should be noted that the ${}_{10}C_2$ was able to train with all of the digits; where as no single weight matrix was able to train all of these digits due to memory limitations. So, from that perspective, even the 93% training set classification was okay..

When the ${}_{10}C_2$ classifier was trained against the smaller set, allowing for an untrained testing set, it did very poorly classifying the training set ($\approx 82\%$). However, when classifying the untrained digits, it performed the best of any of the matrices trained, correctly classifying ($\approx 84\%$), 5% higher than the runner up.

0.5 Conclusion

Two portions of this research deserve further study. The first is the puzzling drastic improvement of the classifier against untrained digits by reducing the number of features from 960 to 300. It would be interesting to to try different features sets, reducing the number of features to different to targets.

The code was designed to be very flexible as far as the actual feature set used is concerned. In-fact, in addition to $8X5$ and $6X4$ the $30X20$ digits were also compressed to $10X7$. A $70 \times 40 = 2800$ feature set would be interesting to study, especially once it was reduced to 300 or so features. This will undoubtedly lead to memory problems, so it would be helpful to

have a computer with 64 bit architecture (to allow more addressable memory) and several Gigabytes of RAM.

The ${}_{10}C_2$ network also deserves further study. It would be helpful to take a much closer look at exactly what values are produced when a number like an '8' is passed through a classifier designed to differentiate between a '3' and a '7'. Maybe there would be some way to throw out these values, or produce a low confidence. One way might be to train the weight matrices to be trinary instead of binary, so they would produce a '1'-'3'-'unknown' result.

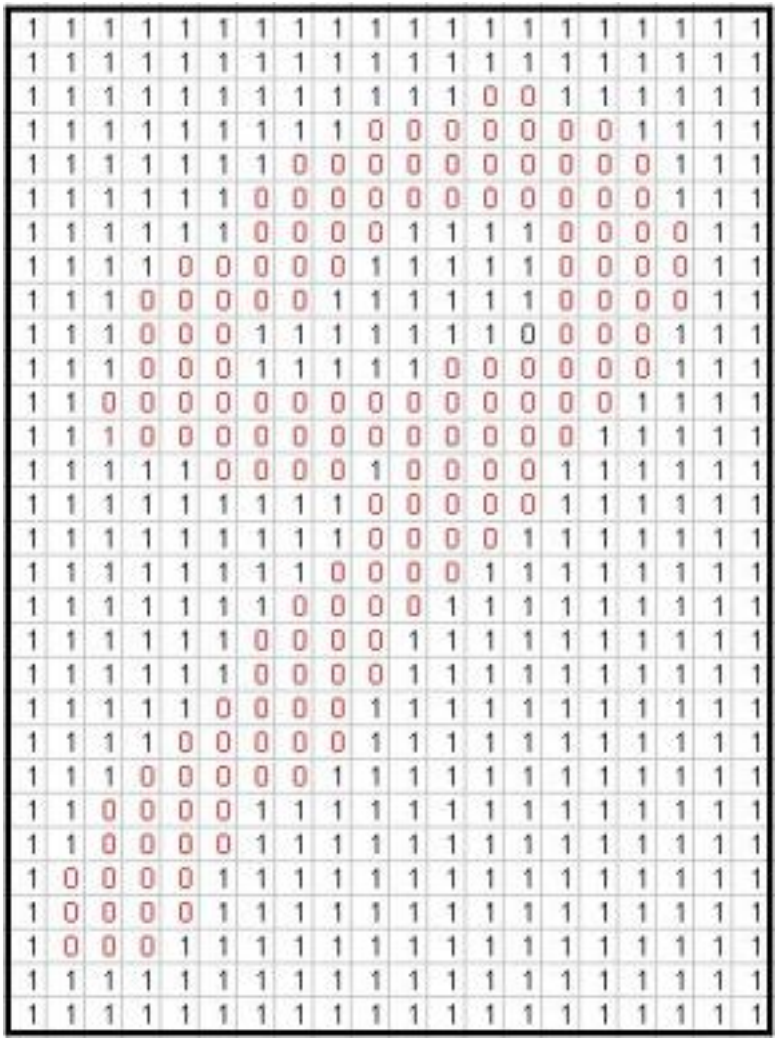

Figure 1: A binary 'nine' as $30X20$ pixels

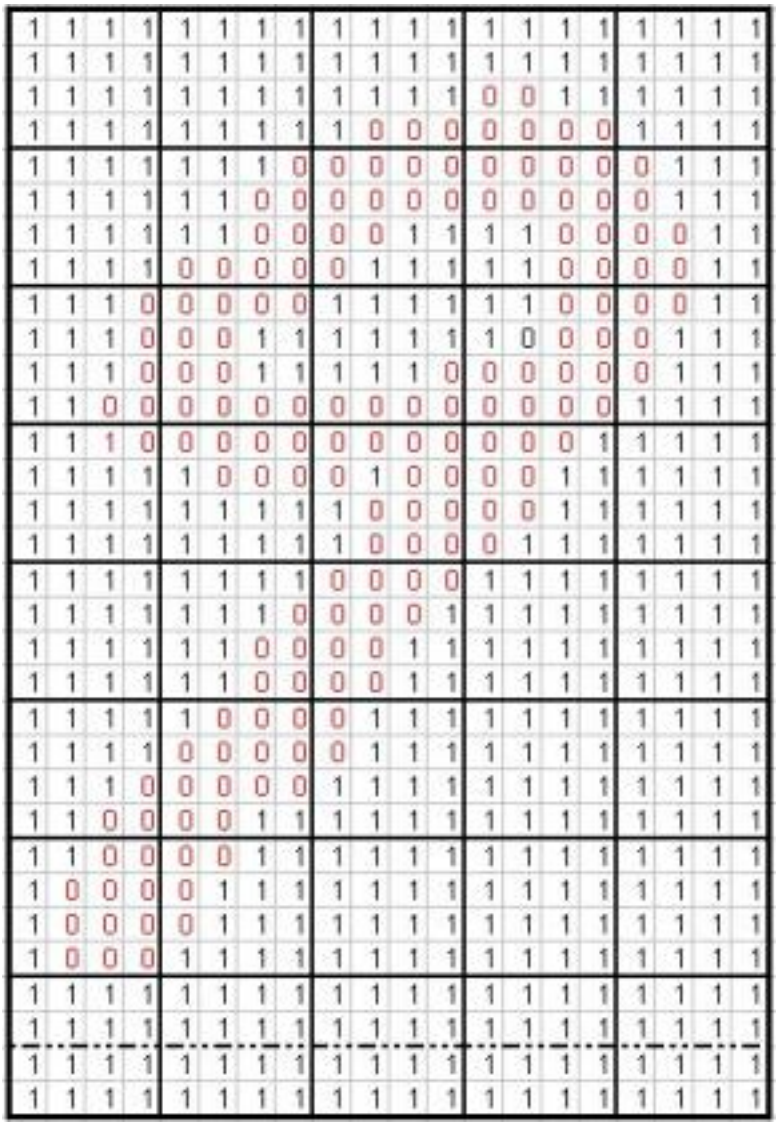

Figure 2: A binary 'nine' padded to $32X20$ pixels and sectioned off into $4X4$ "Fat Pixels"

Figure 3: A 'nine' compressed into a $8X5$ digit

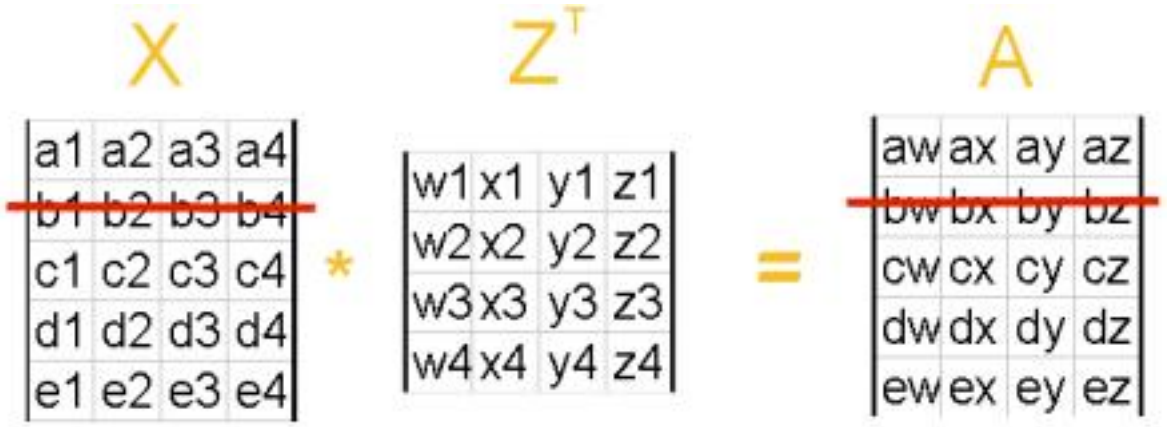

Figure 4: Taking a subset of the A matrix

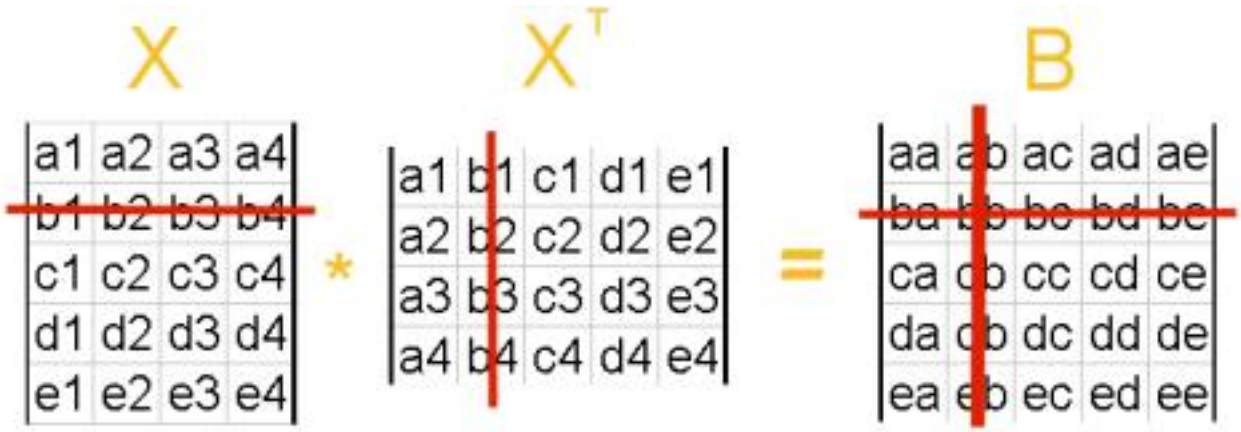

Figure 5: Taking a subset of the B matrix

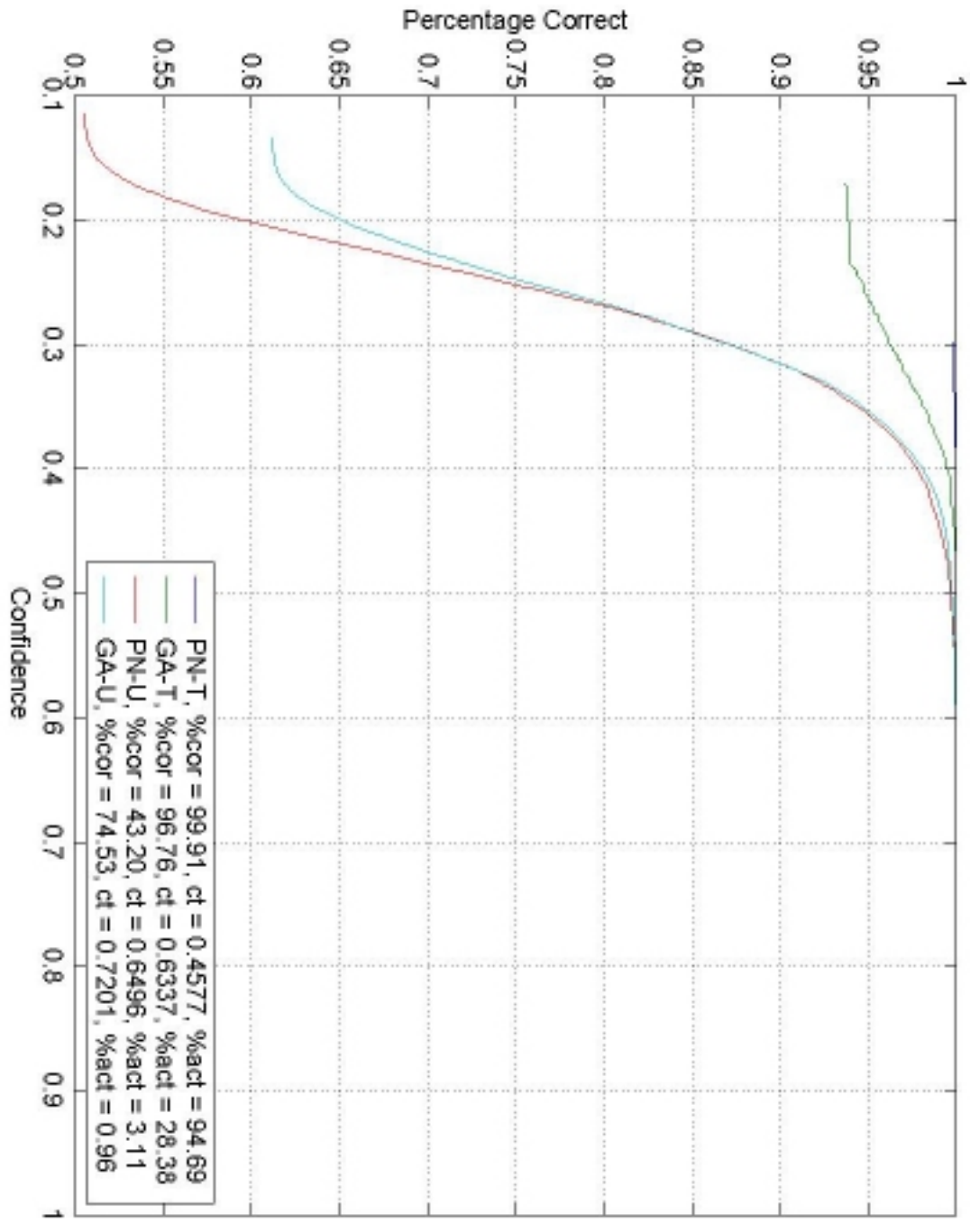

Figure 6: Results of a matrix tested against the training set and 19,173 digits

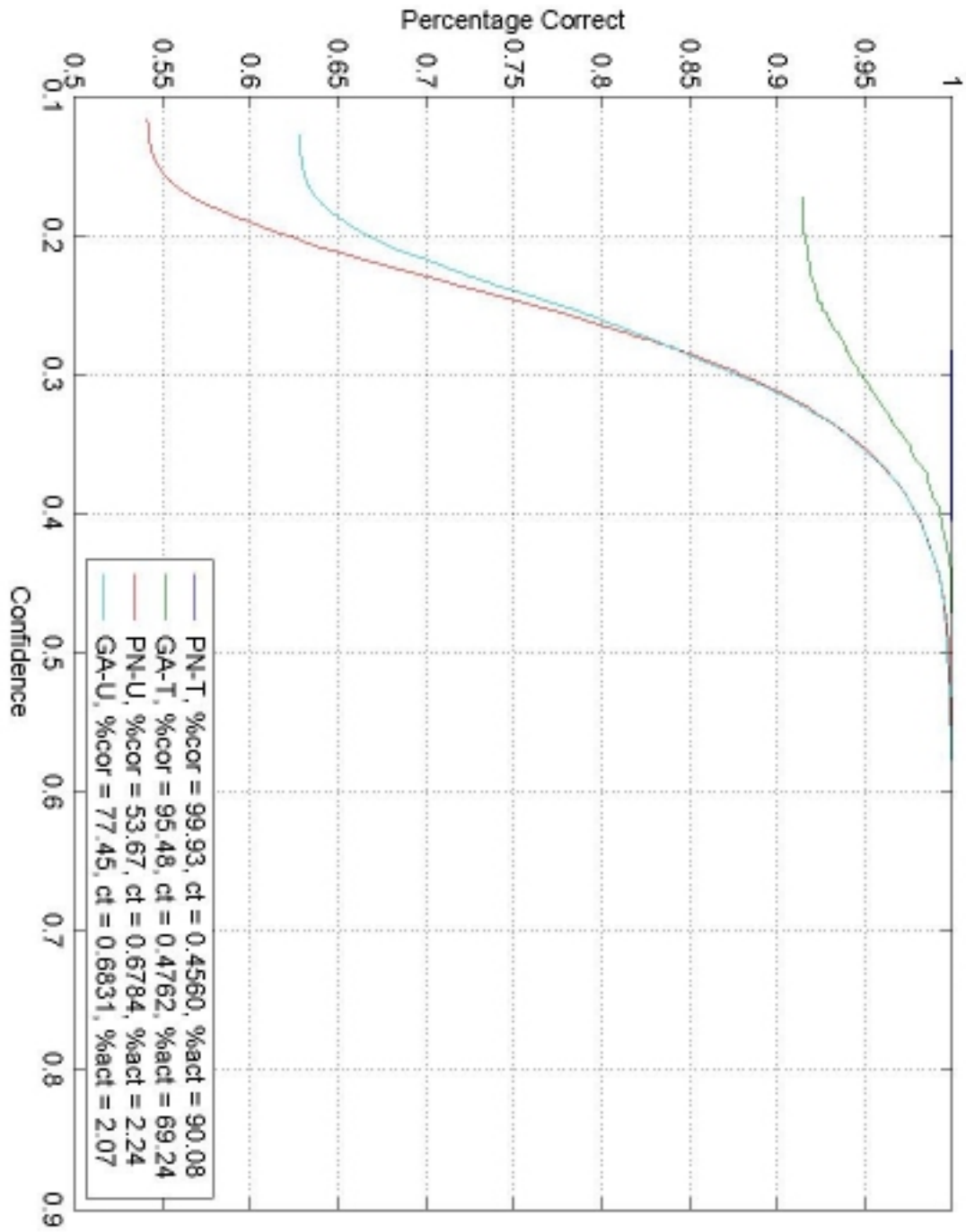

Figure 7: Results of a matrix with a larger training set tested against the training set and 19,173 digits

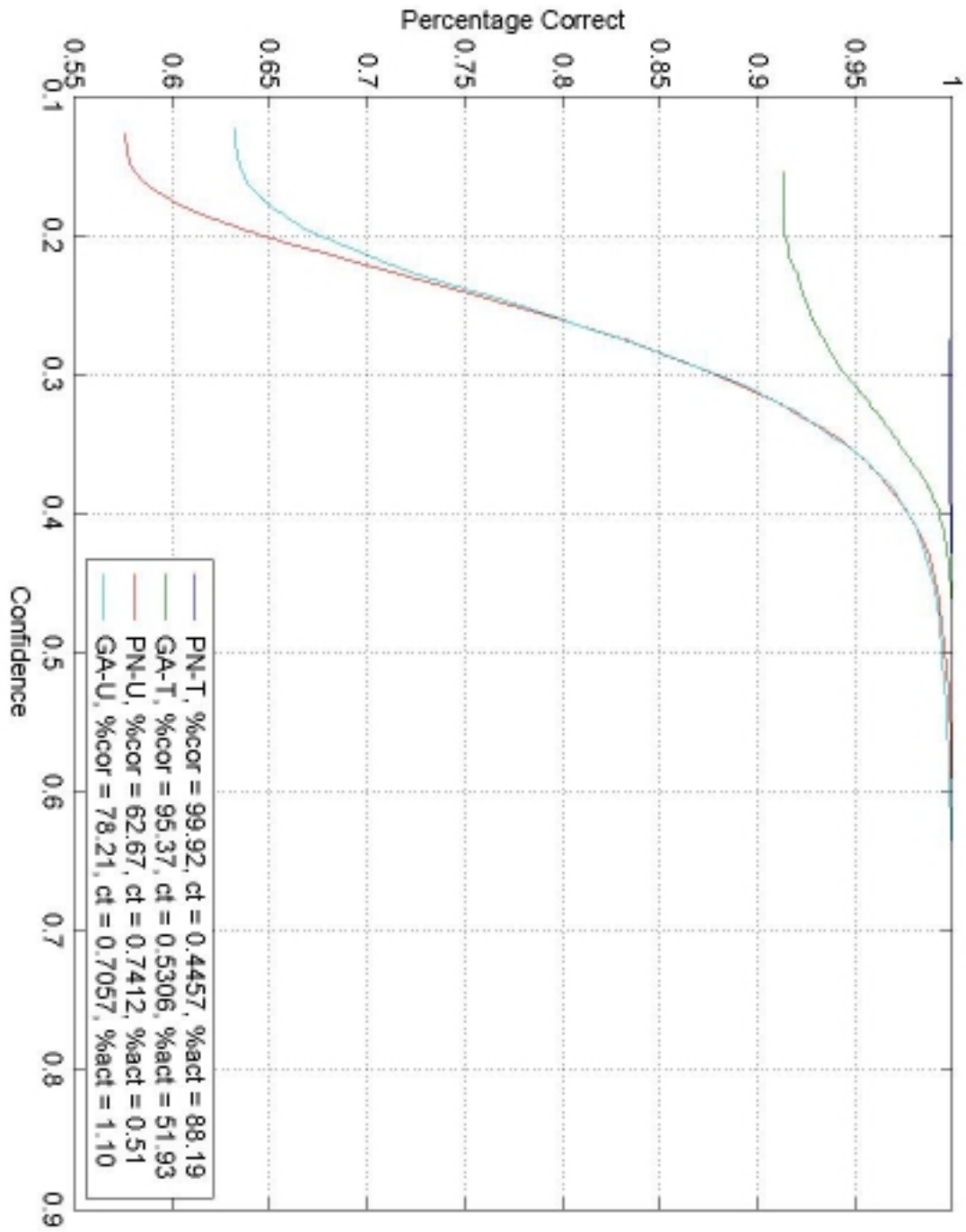

Figure 8: Results of a matrix with a yet larger training set tested against the training set and 19,173 digits

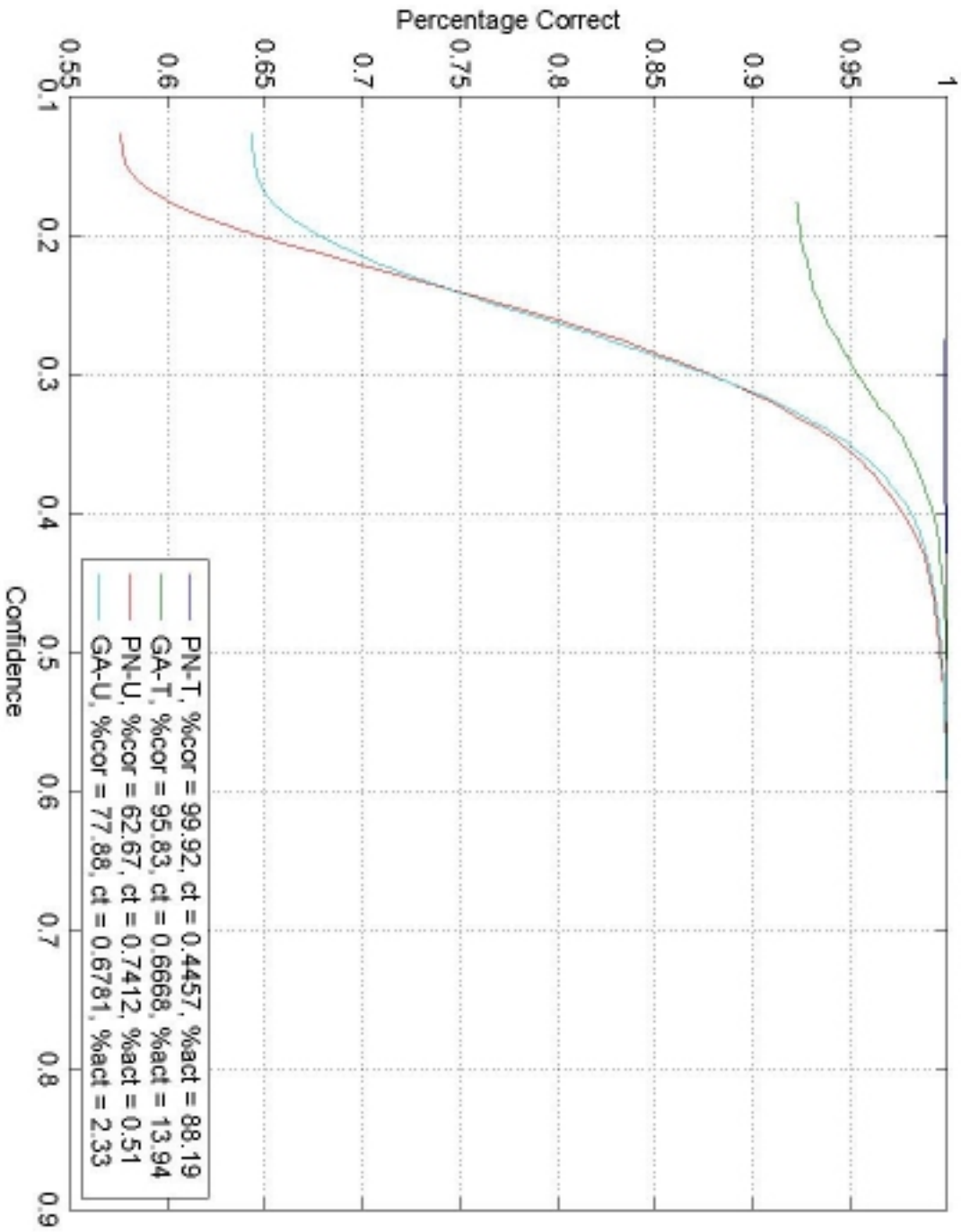

Figure 9: The matrix shown in Figure 8 with a 5 fold increase in GA training iterations

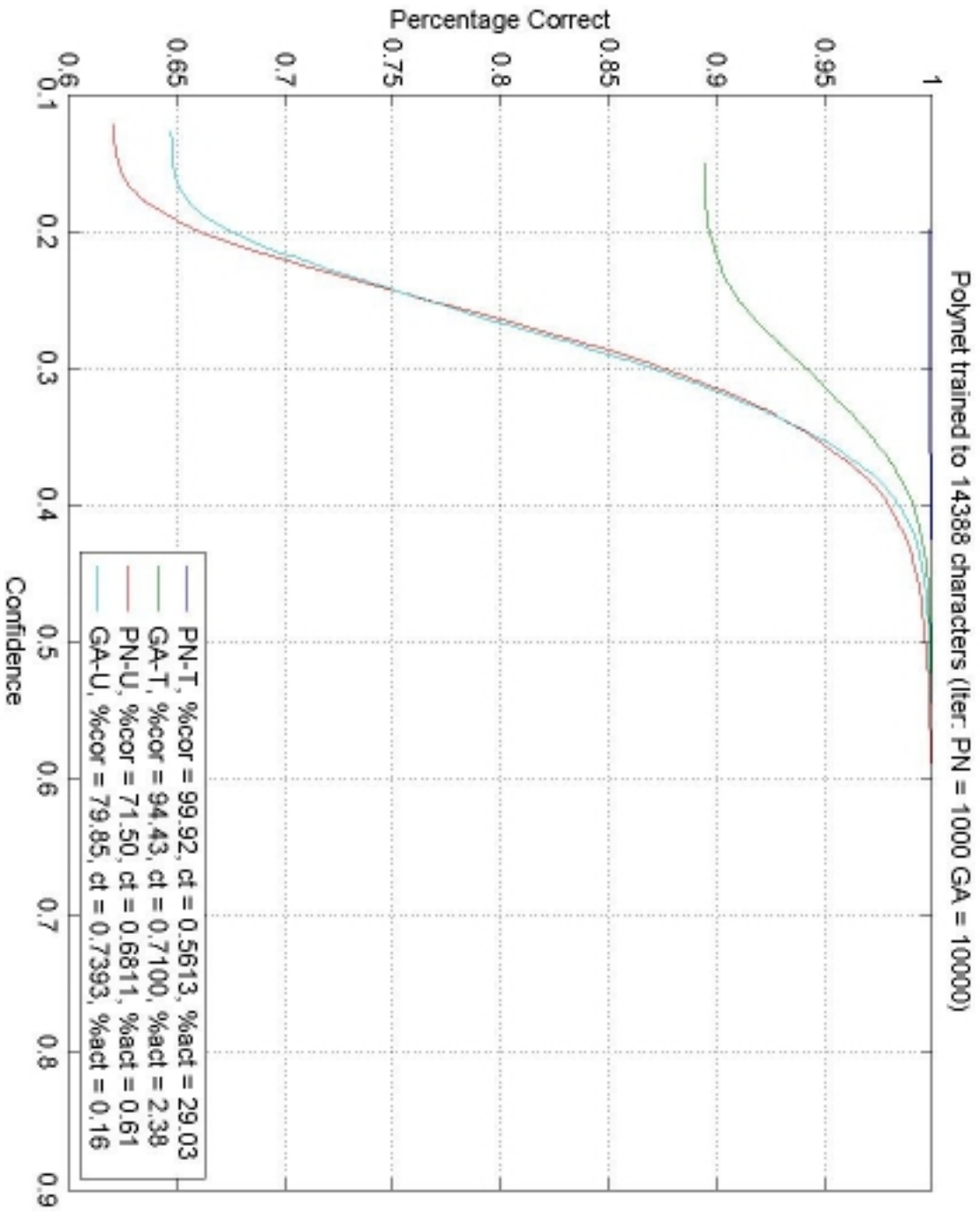

Figure 10: A matrix trained with many digits for many iterations

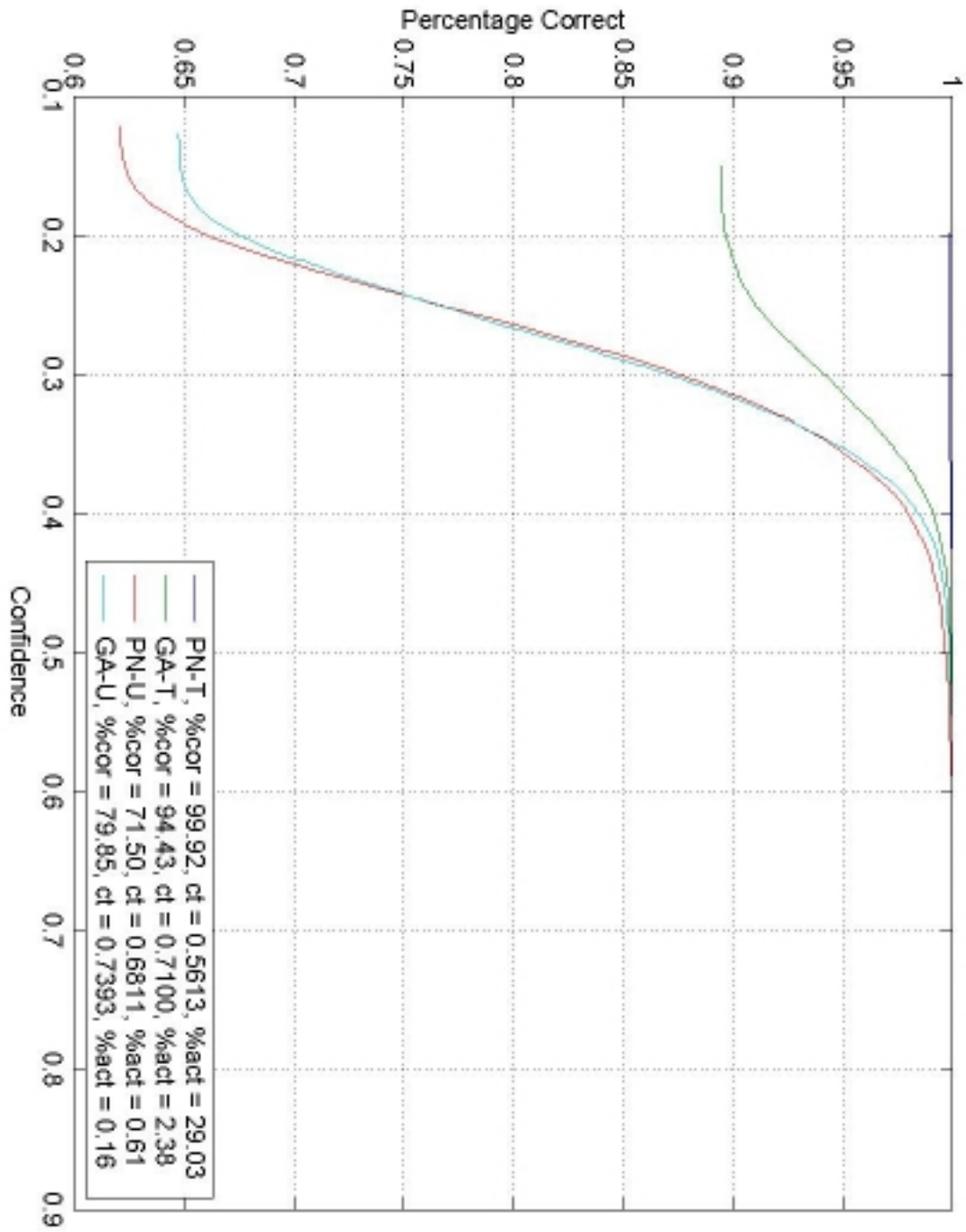

Figure 11: A matrix trained for 3 days to find an ideal 300 features

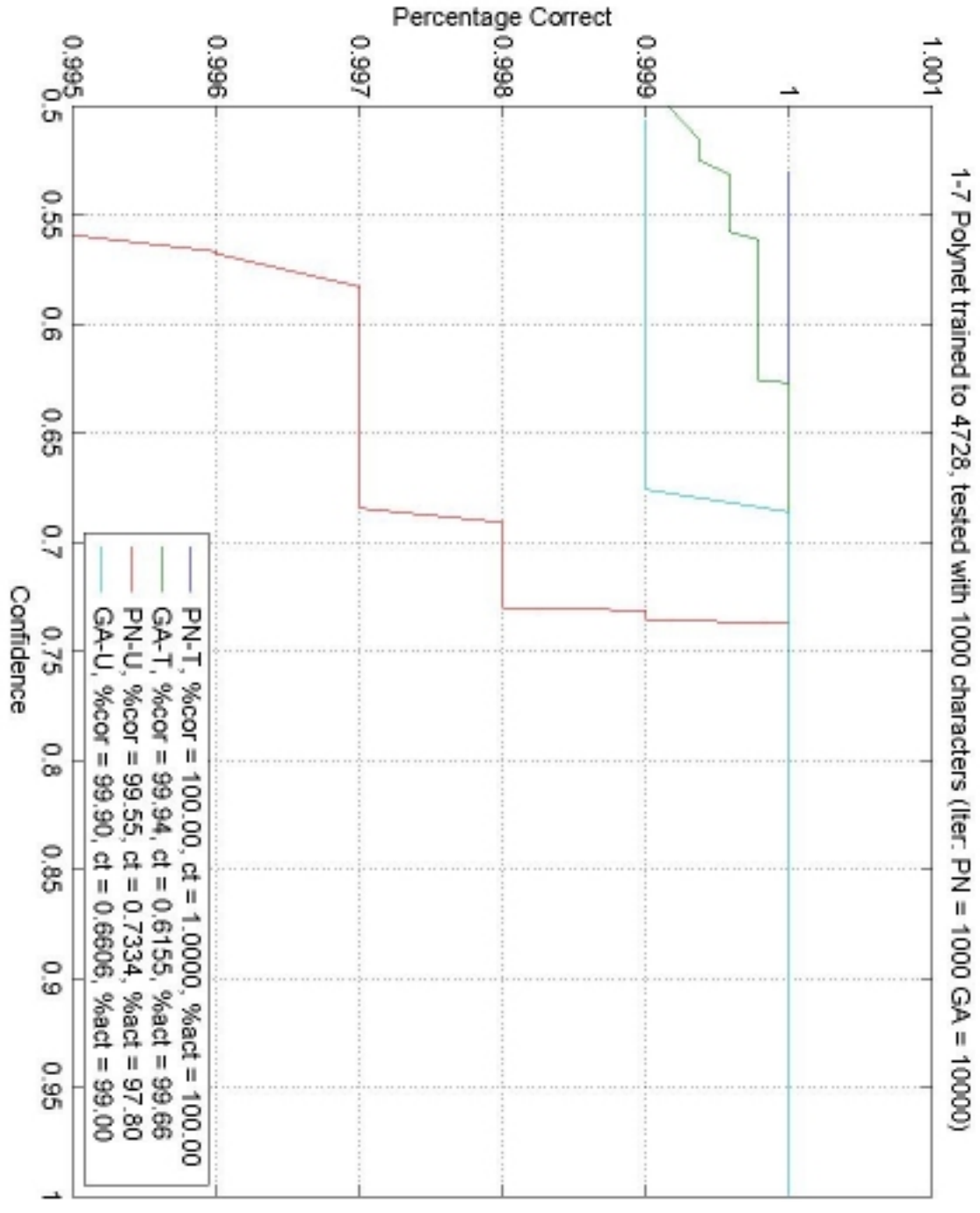

Figure 12: A matrix trained to differentiate between 1 and 7

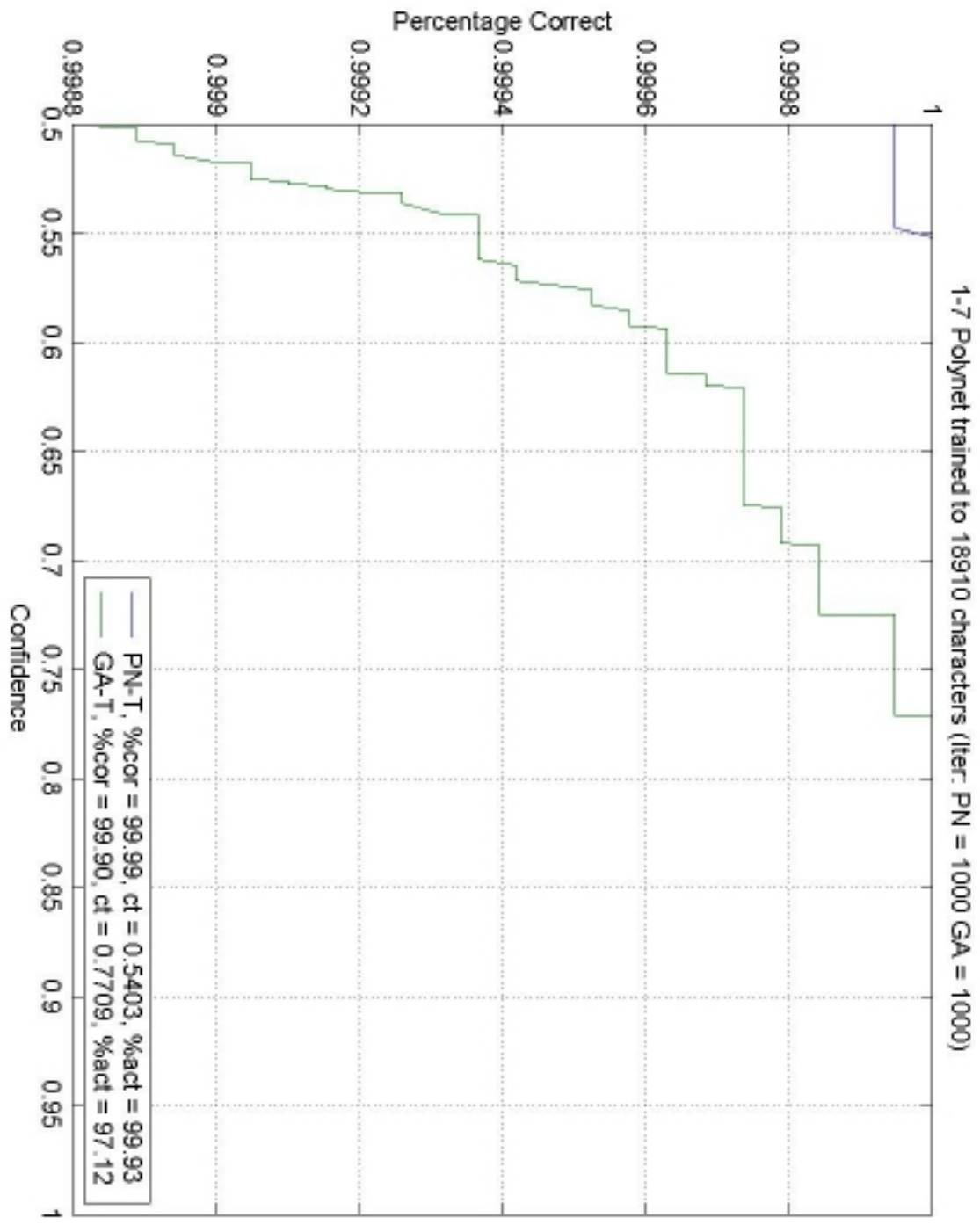

Figure 13: A matrix trained with all 1s and 7s to differentiate between 1 and 7

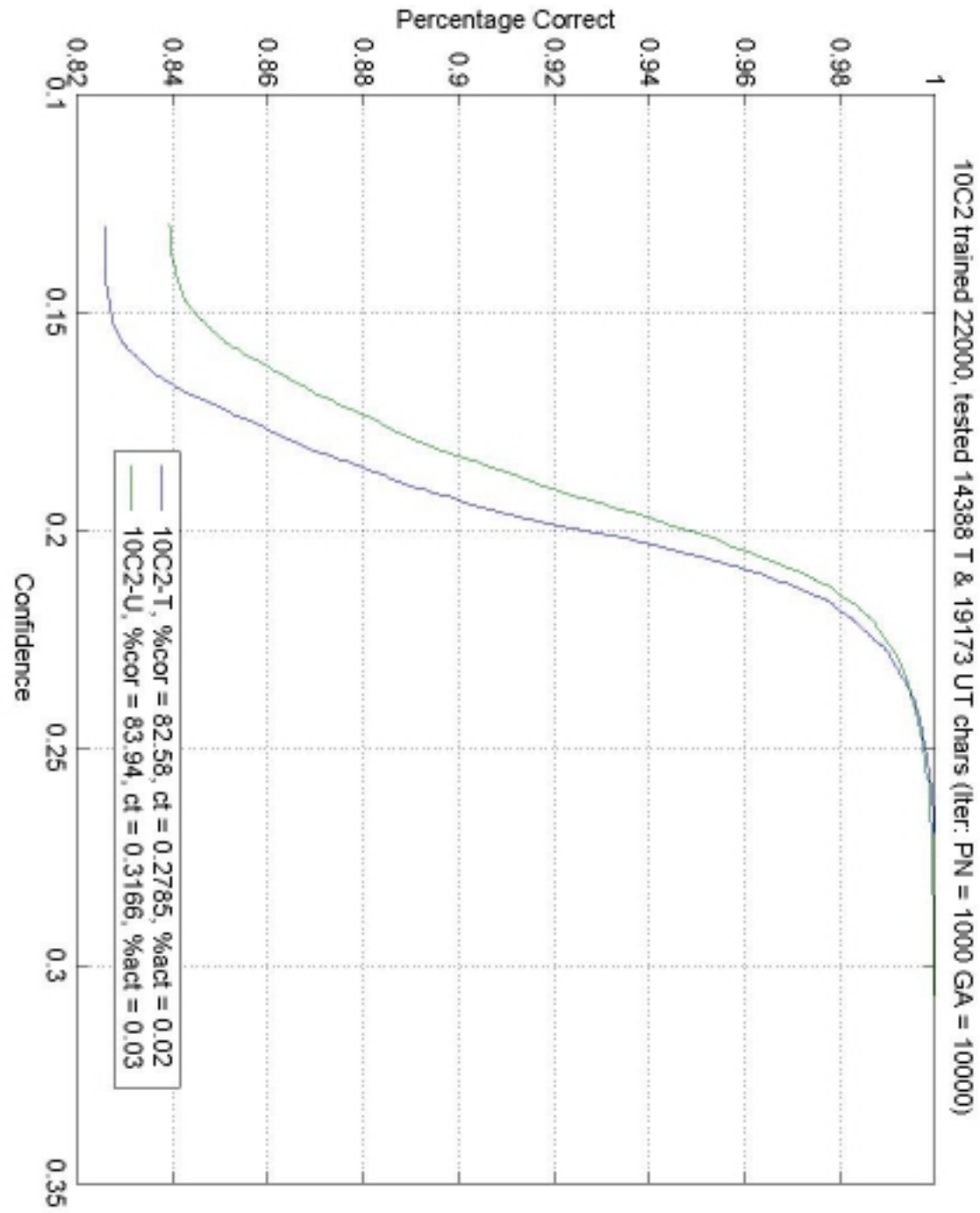

Figure 14: The results of the 10C2 classifier network trained with 25% of the available digits

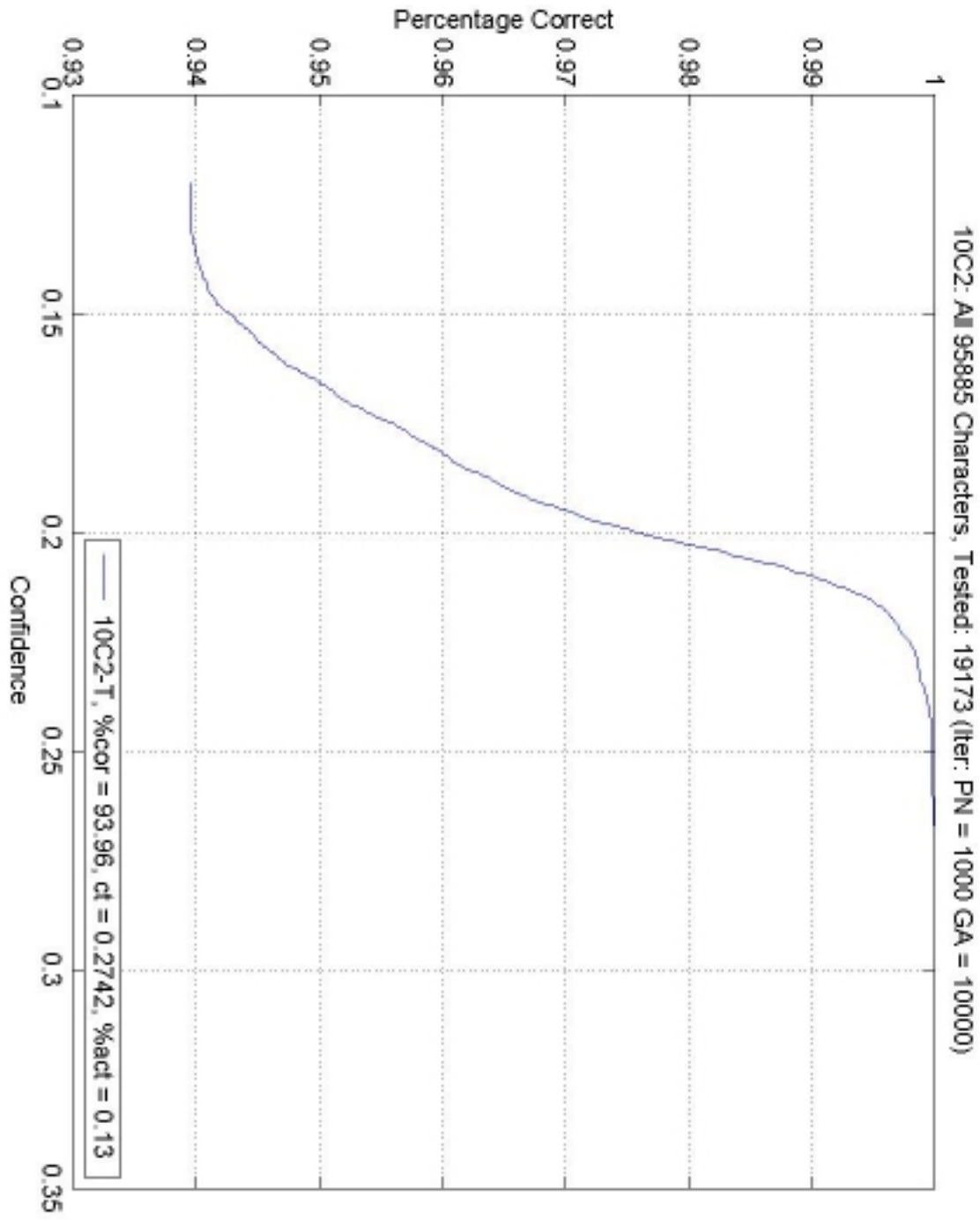

Figure 15: The results of the 10C2 classifier network trained with all 95885 digits

Bibliography

[1] Uma Srinivasan, *Polynomial Discriminant Method for Handwritten Character Digit Recognition,* State University of New York at Buffalo, 1989.

[2] Peter G. Anderson and Roger S. Gaborski, *The Polynomial Method Augmented by Supervised TRaining for Hand-Printed Character Recognition,* Rochester Institute of Technology, 1995 (presented 1993).

[3] Peter G. Anderson, Arun Rao and Roger S. Gaborski, *The Augmented Polynomial Character Recognizer Adapted for Hardware Implementation,* Rochester Institute of Technology, 1993.

[4] Roger S. Gaborski, Peter G. Anderson, David G. Tilley and Christopher T. Asbury, *Genetic Algorithm Selection of Features for Handwritten Character Identification,* Rochester Institute of Technology, 1995.