Defeating CAPTCHAs: Applying Neural Networks

Adam Browning and Dave Kolas

Abstract— For a number of reasons, it is often useful to have a test that can differentiate a human from a computer. These tests, referred to as CAPTCHAs (Completely Automated Public Turing tests to tell Computers and Humans Apart) are very commonly used to prevent computers from registering email addresses and voting in polls on the Web. The most common type of CAPTCHA involves text within an image that is obscured in a number of ways, but able to be read by a human. While several forms of these CAPTCHAs have already been broken, stronger versions are still the primary means of preventing bots from accessing resources on the Web. We endeavor to use a combination of image processing techniques and neural networks to defeat the CAPTCHA used by Hotmail, further proving the need for different techniques for differentiating humans from computers.

***Index Terms*— CAPTCHA, neural networks, image processing**

I. Introduction

There are many sites available on the World Wide Web that intend to limit their resources to individual people. Free email addresses like Yahoo! Mail and Gmail [1, 2], access to tickets from Ticketmaster [3], posting on others' blogs, and voting on polls all fall into this category. In each case, there is enough value in large scale access to these resources that the temptation exists to gain such access. Often, this is attempted through the use of automated software programs specially designed to gain access to these resources, commonly referred to simply as 'bots.' The primary method for preventing the bots from gaining access to the resources in question is employing CAPTCHAs (Completely Automated Public Turing tests to tell Computers and Humans Apart). The simple goal of such a test is to allow valid human users access and deny access to bots.

The most popular type of CAPTCHA by far is based on text recognition. Text is obfuscated in a number of ways, and the human then views the image and enters the text they perceive. The key element is that the image is altered in ways that are easy for a human to interpret but difficult for a computer to interpret. Text may be skewed, rotated, altered in size and color, placed on patterned backgrounds, etc. The result is a wide range of text-recognition CAPTCHAs that vary vastly in difficulty to break.

The goal of our work will be to focus on one particular CAPTCHA, widely regarded as difficult to break due to difficulty in automatically segmenting the image into characters [4]. This is the CAPTCHA Hotmail [5] uses to prevent bots from gaining free email addresses. We will employ a combination of image processing techniques and AI techniques such as neural networks to attempt to gain a reasonable success percentage in breaking the CAPTCHA.

II. Related Work

Related work for this project falls into two primary categories. The first is those who attempted to break CAPTCHAs by applying similar techniques, and the latter is those who have used neural networks for character recognition.

A. Breaking CAPTCHAs

1) Recognizing Objects in Adversarial Clutter: Breaking a Visual CAPTCHA

Clutter in an image complicates the task of determining the words in a CAPTCHA [6]. In Mori and Malik's work, they provide an algorithm for breaking the Gimpy and EZ-Gimpy CAPTCHAs by using knowledge that the words in the image are taken from a dictionary. This approach unfortunately fails to defeat CAPTCHAs where the letters are chosen randomly.

2) Using Machine Learning to Break Visual Human Interaction Proofs (HIPs)

The problem of segmentation, finding the characters, is very simple in many CAPTCHAs [7], reducing them to an OCR problem, which computers are adept at solving quickly. Hotmail's CAPTCHA has been designed to make the segmentation problem more difficult.

3) Using character recognition and segmentation to tell computer from humans

The power of CAPTCHA techniques for deterring automated use of resources lies partly in the fact that their creation is a computationally inexpensive synthesis problem, while solving them is a computationally expensive analysis problem [13]. Further compounding this unbalanced cost is the fact that CAPTCHA producers can modify their algorithms slightly and often require significant and costly changes on the analysis side.

B. Using Neural Networks for Character Recognition

Using neural networks for character recognition has a long history in the artificial intelligence community, and is a well established method.

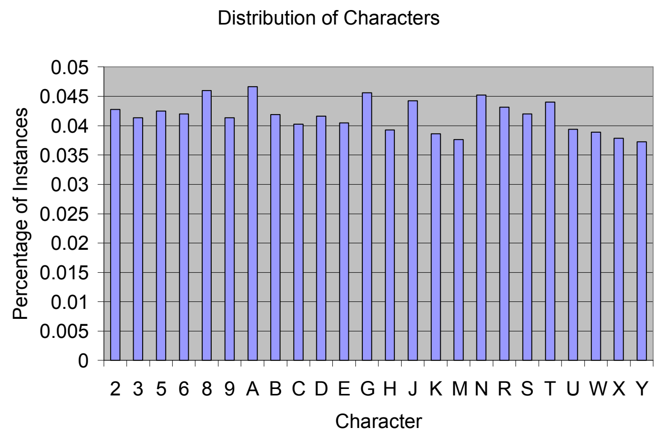
Neural networks have been used in this capacity for a variety of tasks. One of the most fundamental is the task of Optical Character Recognition (OCR). In this task, computer systems attempt to recognize printed letters in order to turn them back into text characters. Many neural networks have been designed for this task [8], with much success [9].

Neural networks have also been used for tasks related to OCR. They have been used to recognize the characters that are input into a touch terminal [10]. They have even been used for OCR on Chinese characters [11].

III. Training Data Gathering

We have gathered a training set of CAPTCHAs from the Microsoft Hotmail service. We have captured and manually decoded 1000 unique CAPTCHA images. These are divided into two groups; the first group is used as training data for the recognition algorithms. The second group will then be used to test the recognition of the first group.

Each CAPTCHA image is comprised of 8 characters. Each of these characters is taken from a set of 24 characters, a subset of the capital letters and digits created by removing ambiguous pairs of characters from the set. Each of the 24 characters is distributed approximately randomly within the set, as shown in the following diagram.



With 300-400 examples of each character, there should be enough data to suitably train and test recognition of each character.

IV. Approach

The approach to breaking this CAPTCHA has two major components. The first component is a series of image processing steps that attempts to deal with the noise in the image, the extraneous lines, and divide the image into images for each character. The second component attempts to recognize the characters contained within each image. Each of these components requires different experimentation in order to appropriately tune it to the problem at hand. In the end, the two stages are combined together to attempt to break the CAPTCHA.

Our experiment attempts to extract the characters from CAPTCHAs with an accuracy rate of 1% or better. This test combines the error rates of every step along the way.

V. Image Processing

Our approach to breaking this CAPTCHA is to gather the lines in the image into connected chunks, which will be kept or discarded based on an acceptance function that considers the height and width of the block. Once the image has been pre-processed to remove extraneous clutter, we believe the neural network will be significantly more effective at breaking the CAPTCHA.

A. Binarization

To prepare the images for pre-processing, we convert from grey-scale JPEG formatted images to monochrome PNG formatted images using a threshold function to determine which pixels to store as black, and which pixels to store as white. The image handling libraries in Java use an additive model, packing the red, green, blue and alpha channels of each pixel into a single integer value. To determine if a single pixel should be stored as black or white, the individual red, green, and blue channels were retrieved and added together. The threshold value to determine if a pixel should be black or white was determined through trial and error to be 440, as a more stringent threshold resulted in significant character losses, whereas a less stringent threshold yielded excessive noise. Figures IV-1 and IV-2 below show a sample image before and after application of this function.

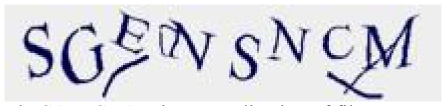


Figure V-1: Sample CAPTCHA prior to application of filter

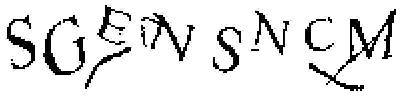
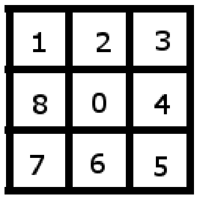


Figure V-2: Sample CAPTCHA after application of filter

B. Connected Points Detection

The second stage of filtering is to determine which points in the monochrome image are connected. To enhance performance at this stage, only the neighbor directly to the right and the three neighbors below a point are considered for connectedness. Visually, if pixel 0 in the graphic below is the one currently being considered, we examine pixels 4,5,6 and 7 and add them to the connected list if they are set to black.



1	2	3
8	0	4
7	6	5

To further simplify this problem and reduce the memory footprint of this stage, we utilize the ability to alias objects in Java to avoid storing the same list multiple times. This approach avoids having to perform a join of lists later in the algorithm. One area in which the performance of the current implementation of this stage could be improved is in the storage of the lists. Currently we store the lists in a hash map, keyed by the pixels, whereas they could be stored in a one-dimensional array whose coordinate is calculated as row * width + column. Performance improvements of the naïve implementation of such a technique would be minimal, however most of the calculation can be pulled out of the loop and three multiplications, one subtraction and one addition replaced by three addition operations.

C. Noise Reduction Filtering

Once the image has been converted to monochrome and connected points discovered, filters are applied to discard non-connected noise lines.

The first level of noise filtering applied to the monochrome image removes glyphs that are not crossed by a band running from four pixels above the vertical midpoint of the image to two pixels below the vertical midpoint of the image. Although of limited power, this filtering stage can be executed rapidly, and does remove some of the noise arcs. The two primary benefits of this are that it allows future filters to deal with less noise, while reducing the amount of junk input to the neural network. Figures IV-3 and IV-4 below show the same image before and after application of this filter.

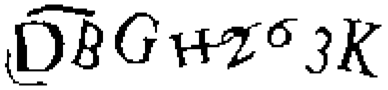


Figure IV-3: Sample CAPTCHA before application of noise removal filter

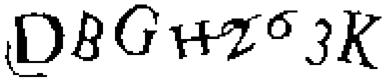


Figure IV-4: Sample CAPTCHA after application of noise removal filter

The second stage of noise reduction filtering proceeds by calculating the height and width of the remaining sets of connected points. If the size of the set falls below a specific threshold, the set is discarded. The efficacy of this phase of filtering was greatly reduced by the widely varying sizes of glyphs within the CAPTCHA. As a result of some of the glyphs being as small as six pixels wide and twelve pixels high, the threshold could be set no higher than that without losing characters.

D. Segmentation of Connected Glyphs

The final stage of preparing the image for character recognition is to attempt to segment glyphs connected by a noise arc into separate characters. This phase of the filtering proved in many cases to be a significant challenge, as was expected from the results of Chellapilla and Simard [7].

For images that are not connected by noise arcs, segmentation of glyphs in a monochrome image is a simple task, requiring nothing more than finding connected points, although separating noise glyphs from actual characters is more difficult. An example of the results of performing this segmentation is below.

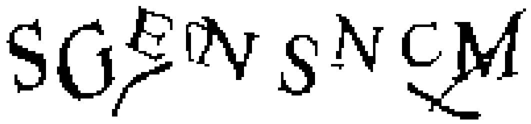
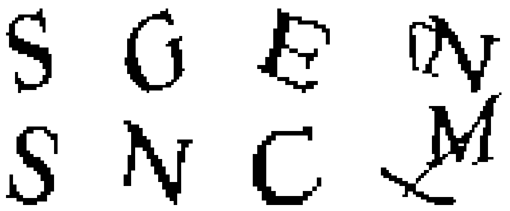


Image Prior To Segmentation



Unfortunately, as evidenced in the first N and M above, noise arcs are still present in the segmented images. Character recognition algorithms may be able to recognize the characters even in the presence of such noise, but image processing techniques may be capable of reducing the noise further. The noise reduction techniques that could be applied in the timeframe of this research were of limited effectiveness; however the future work section of this document discusses more options that we believe will yield improved results.

One factor simplifying this portion of the filtering is that it is known *a priori* that there would be exactly eight glyphs contained within the image. With this information, we can detect the case where two or more characters are either connected or one or more have been discarded in earlier phases of filtering. Due to the previous filtering phases erring on the side of not discarding a set of connected points, we assume that reaching this point with fewer than eight sets of connected points implied that two glyphs were connected by a noise arc.

In the event that fewer than eight glyphs were detected, we recursively selected the widest glyph and applied splitting algorithm to that glyph, adding the two or more split lists back to the list of glyphs until eight glyphs were detected.

The splitting algorithm employed used a threshold function whose acceptance threshold was calculated using the weights of the columns using the following algorithm.

```
For y in rows
  For x in columns
    If pixel(x,y) == black
      Weight[y] -> weight[y] + 1
    End if
  End for
End for
```

Once the weights were found, the average and standard deviation of the weights were calculated and the threshold was set to the average minus the standard deviation. This method performed very well in cases where the standard deviation was less than half the average. An example of a glyph meeting this requirement is Figure IV-5.

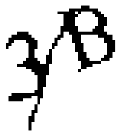


Figure IV-5: Glyph With Multiple Connected Characters

Calculating the weight, average and standard deviation of Figure IV-5, we get the following, where the X-axis is the column of the image.

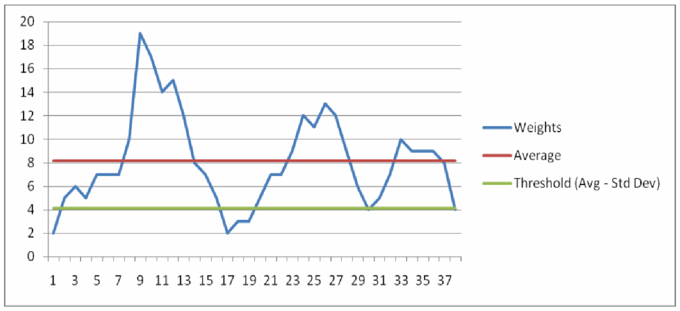


Figure IV-6: Statistical Information for Figure IV-5

Using the calculated average and standard deviation to split the characters results in the image shown in Figure IV-7, which would then be split into segments for recognition.

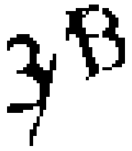


Figure IV-7: Results of Applying Splitting Algorithm to Figure IV-5

The performance of this algorithm suffered greatly, however, when faced with images where the standard deviation of the weights was nearly as great, or greater, than the average of the weights. An example of a segment demonstrating this effect is shown in Figure IV-8.

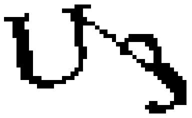


Figure IV-8: Segment Resistant to Splitting Algorithm

As shown in Figure IV-9 below, the threshold for this image is smaller than any of the weights presented in the segment.

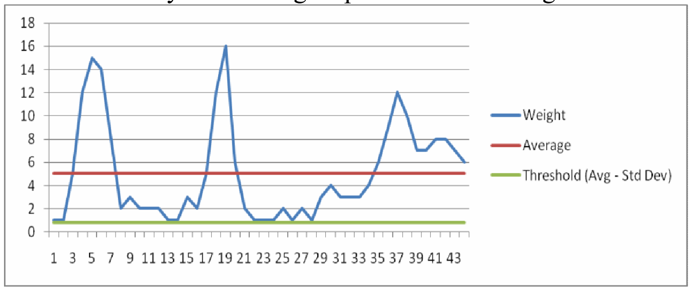


Figure IV-9: Statistical Information for Figure IV-8

Unfortunately, due to the narrow stroke used in portions of the *U* in Figure IV-8, the obvious algorithm of splitting at the lowest weight fails. Attempting to split at the narrowest points yields the glyphs shown in Figure IV-10 below.

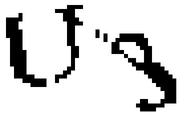


Figure IV-10: Results of Splitting Figure IV-8 at Lowest Weights

Although the image in Figure IV-10 looks like a successful split, the following segmentation phase would result in five separate glyphs, of which two could be discarded based on size. To more easily visualize the results of the following segmentation phase, Figure IV-11 below splits the image with red bars rather than the background color.

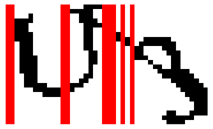


Figure IV-11: Results of Splitting Figure IV-8 with Red Bars at Lowest Weights

In the particular case of Figure IV-8, it may be possible to prevent splitting in the individual characters by examining neighboring pixels, however this was not tested as part of this study.

E. Image Processing Performance

Performance of the image processing routines was within performance goals set for this study, with an average time of 785.7 milliseconds per image. Even with the image processing's current acceptable performance, mounting a real-world attack would require further performance enhancements. Moving from using the high-level methods of Java's BufferedImage class to the low-level methods of the DataBuffer class should improve performance dramatically. Additionally, modifying the method that makes the image monochrome to turn the image into a one-dimensional array of Boolean values should provide further performance improvements during latter phases of image processing.

VI. Using the Neural Network for Character Recognition

The character recognition portion of the CAPTCHA recognition will be performed using a neural network. To use a neural network without spending the effort on writing neural network software, we make use of the neural network software package JOONE [12].

A. Neural Network Design

For our purposes, we decided to follow some of the OCR precedent in terms of our neural network design. Because the problem task is essentially stateless, we used a feed-forward neural network. This means that after training, the results of applying the neural network to a character image are not needed for the network to function in the future. We chose a neural network design with one hidden layer, primarily due to a combination of the need to reduce the number of variables with which we were experimenting and the fact that the multi-hidden-layer neural networks took a fantastically long time to train. Thus the neural network has three layers. The input layer is a direct interpretation of the pixels of the image, rendered in black and white. This means that the size of the first layer is dependent on the image resolution that is used in the particular experiment. The second layer is a hidden layer of varying size. Through the course of the experiments, we used several different sized hidden layers to compare results. This layer utilizes the sigmoid function to calculate its outputs. The final layer is also a sigmoid function layer. Its size corresponds to the number of characters that we are attempting to recognize. We will expect the values of these outputs to range from 0 to 1, depending on how much the input is recognized as each of the characters. Back-propagation was used to train the networks.

B. Tuning the Neural Network

In order to make use of this system, we need to choose ideal values for each of the parameters of the network.

The first choice that needs to be made is the number of hidden layers that will be used in the neural network. For these experiments, we chose only one hidden layer as this seems to be common in the literature and provide the power necessary for the character recognition task.

The second parameter that needs to be set in order for the neural network to be trained is the learning rate. If one conceptualizes the varying weights of the links between the nodes of the neural network as a multidimensional surface, the learning rate corresponds to the velocity at which the learning moves the current status around the surface.

The third parameter that must be set is the momentum of the training. Using the previous conceptualization of the learning as a multidimensional surface, the momentum corresponds to the momentum of the current value as it moves across the surface. Having momentum prevents the learning from oscillating back and forth without utility.

Fourth, the size of the middle hidden layer must be set. The ideal size of this layer strikes a balance between being complex enough to recognize the training data and the validation tests and also will not overtrain on the training data.

Finally, the resolution of the images that are used can be varied. In some sense, this parameter is the most difficult to predict the effect of, because one can see the advantages of both high resolution and low resolution images. With respect to the high resolution images, more data is available to the neural network to train on. However, with the lower resolution images, the neural network sees the data at a "higher level", possibly allowing it to generalize better to new examples.

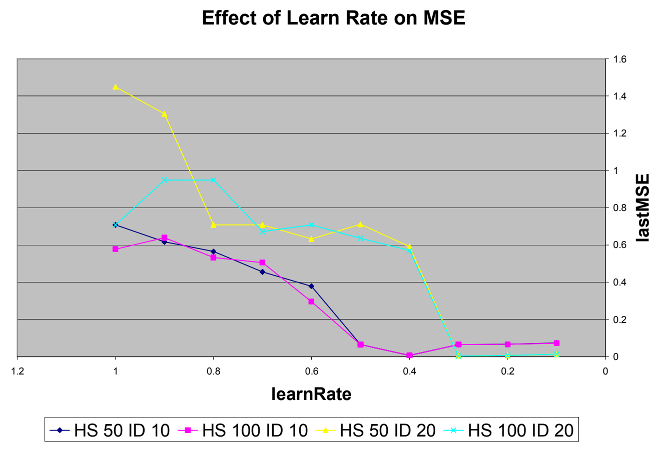
As a result, we design an intermediate experiment to help determine how these parameters affect the learning of the neural network.

1) *Neural Network Parameters Experiment*

We hope to design an experiment that can provide insight into the ideal parameters for the neural network. To do this, we use several experiments over a sub-problem of the character recognition that will be needed for identifying the characters in the CAPTCHAs. As a sub-problem, we used recognition of the numeric characters 0-10. The goal of these experiments will be to get a sense for how varying these parameters affects the convergence of the network.

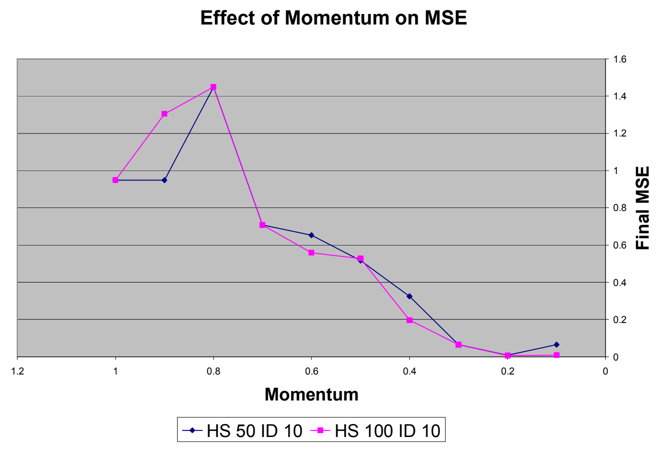
The first thing that the experiment will need is a set of training data. In order to easily acquire a large set, we will generate it. Using the font renderer built into Java for screen and image font rendering, we generated twelve examples of each of the ten numbers in an 80x80 pixel image. For the examples, we used twelve readily available fonts ranging from Times New Roman to the much thicker Impact. This set provided a basis for us to see how the various parameters affect the mean-squared-error (MSE) on a training set.

A first experiment showed how the learning rate affects the MSE of the neural net on the training set. Here, we used data for both a hidden size of 50 and 100 nodes and an image resolution of 10x10 and 20x20. For this test, the momentum was fixed at 0.2, and the number of learning cycles was fixed at 1000. The following graph shows the results of these experiments.



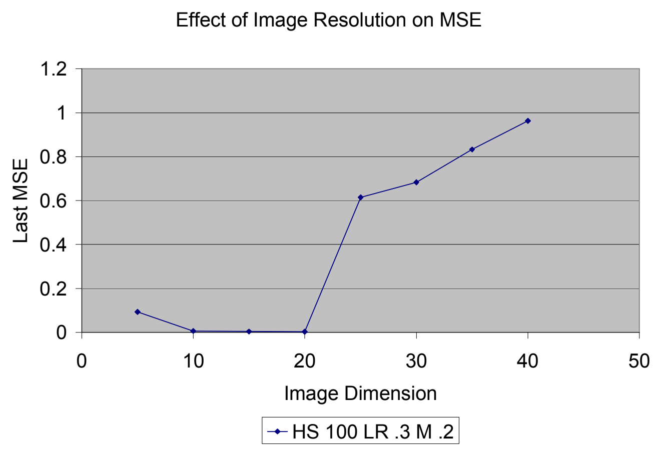
This graph shows the final MSE plotted against the varying learn rates. It shows that generally, as the learn rate decreases, the resulting MSE is also reduced. Interestingly, the MSE actually starts to go back up towards the lower learning rates, but this is likely because they had not reached their minimum stable values during the 1000 cycles. From this, we learned that a lower learning rate is superior for minimizing the total MSE when the amount of time required for convergence is not a problem.

A second experiment shows the effect of momentum on the resultant MSE. We show data for both a hidden layer size of 50 and 100 nodes, with a fixed resolution of 10x10 and a fixed number of cycles as 1000. The learning rate is also fixed at 0.4. The following graph shows the result of the experiment.



From the graph, we can see that when the momentum is closer to 1, the MSE does not reduce much at all though the course of the 1000 cycles. As the momentum decreases, the resulting MSE starts to decline. However, it starts to go back up slightly in the case of the 50-node hidden layer. From this, we know that we should start with a low momentum for the neural network.

A third experiment deals with the varying input resolution of the images. It is not necessarily easy to guess whether higher or lower resolutions will provide better results. On the one hand, the higher resolution images provide more input information to the neural network, and thus the neural network has more specific data to work with when determining the characters. On the other hand, using a lower resolution allows the neural network to view the problem more abstractly, and may result in easier convergence of looking at the various different letters. In order to get a better idea of how the image size affects the convergence of the network, we ran an experiment that varied the size of the input image while holding the hidden size, learn rate, and momentum fixed. With the hidden size set at 100, the learn rate set at 0.3, and the momentum at 0.2, the image dimensions were altered from 5x5 to 40x40. This yielded the following graph:



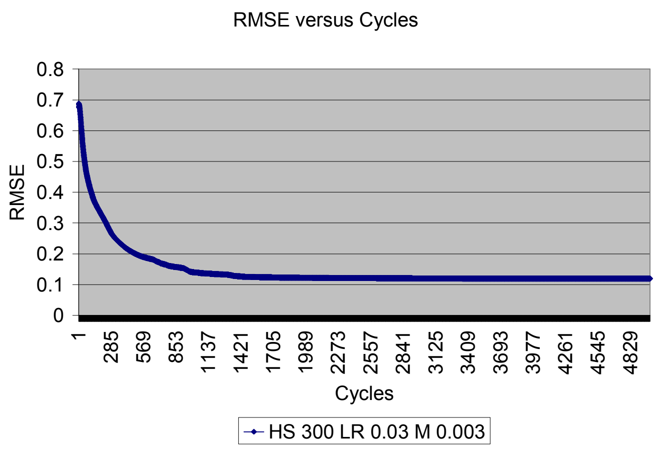
When the image dimension was 5x5, the resulting Root Mean Squared Error on the training set was close to 0.1. This indicates that while the neural network was able to reasonably converge, it was not able to get enough information from the digits to distinguish them fully. When the input image was 10x10, the neural network was able to do significantly better at reducing the end RMSE. This makes sense, as the neural network had four times as much information from which to derive the digits. The trend continues downward through 20x20, at which the final RMSE was approximately 0.003. However, after 20x20, we see a dramatic change in the performance. It appears that the resulting image essentially becomes too "noisy", and the neural network does not converge to low levels at all. As the image size continues to increase, this trend continues. When the image size reaches 40x40, the neural network is unable to learn to do any better than merely guessing.

From these experiments, we learned some general concepts about tuning the neural network for character recognition. First, we need a fairly small learning rate in order to account for the complexity of the input data, especially with the 20x20 images. Second, keeping the momentum reasonably small as well allows the neural network to delve towards the global minimum error. Third, we need to choose a somewhat middling image resolution in order to both give the neural network enough information to intelligently make its decisions and yet not give it so much information as to be overwhelmed.

C. Training for the Actual Data Set

Training the neural networks for the actual data sets was much more time consuming. Instead of the ten digits, the neural network had to be trained on the full set of 24 digits and capital letters used by the CAPTCHA. Again, the training set was generated using the Java font rendering engine. The training set made use of 12 fonts, and an 80x80 image was generated for each character in each font. This resulted in a total of 288 training images. The larger training set was a major factor in increasing the amount of time taken to train.

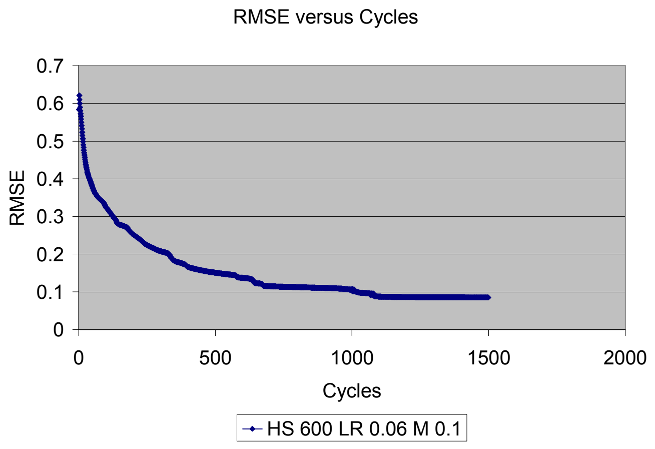
Because of the complexity of the data set, we chose to start with a very low learning rate (0.03) and low momentum as well (0.003). The number of training cycles was increased from 1000 to 5000. A hidden layer of 300 nodes was used for this experiment. The following graph shows the progress of the net's training though the cycles.



As we can see from this graph, while the RMSE is slowly decreasing, it appears to by asymptotically approaching a RMSE greater than 0.1. This does not seem to be a particularly good value, so we tried to alter the parameters somewhat.

For the next test, we increased the size of the hidden layer to 600 nodes. The learn rate was increased from 0.03 to 0.06, and the momentum increased to 0.1. The goal of increasing the momentum was to prevent getting stuck in local minima. Looking at the above graph, since not much notable improvement was seen after 1500 cycles, we decided to reduce the number of cycles from 5000 to 1500.

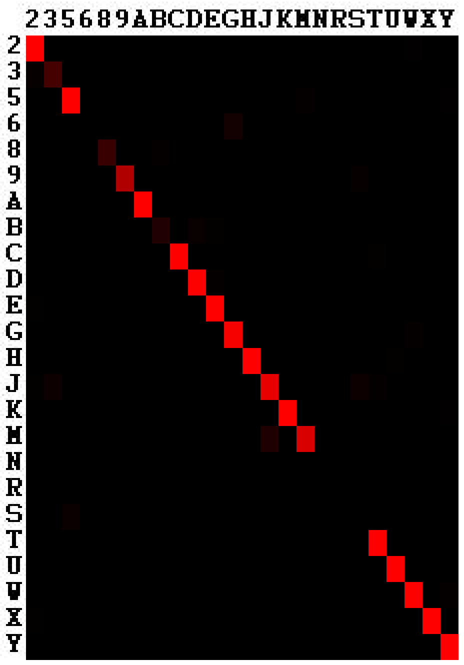
This resulted in the following graph of RMSE versus cycles.



This graph shows the neural network working its way downwards below 0.1. Thus the increased hidden size allowed the neural network to achieve lower error rates on the training set, though only a small improvement. These nets were then used to attempt the first recognition on the unknown font and segmented images data sets.

1) Testing on the Unknown Font

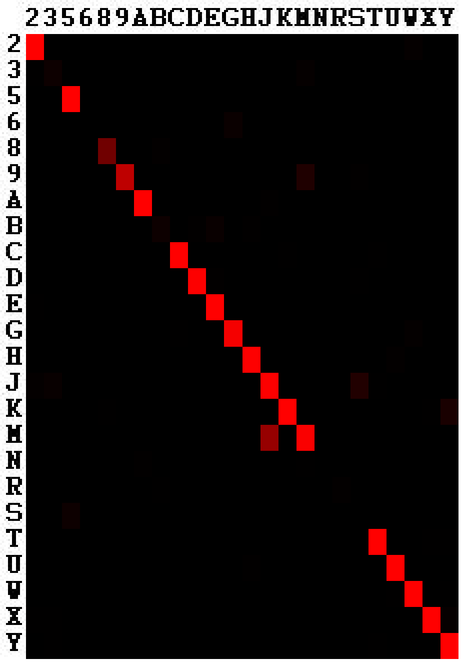
Along with the 24 characters in the twelve training fonts, a 13th font was generated with all 24 characters for testing. Each of the training trials was validated against this set of images when they finished. For the first test, the validation correlation looked like this:



Correlation for LR 0.03, M 0.003, HS 300, 5000 cycles

The rows in this diagram represent the input letters, and the columns represent what they were identified as. For example, the letter M here was identified strongly as an M, but weakly as a J. Interestingly, most of the errors seem to involve the letter not being recognized at all.

The correlation on this data for the second net does not look significantly different:



Correlation for LR 0.06, M 0.1, HS 600, 1500 cycles

In fact, this diagram actually shows that this net, despite its better RMSE on the training set, performs more poorly on the validation set. Looking closely at this diagram, we can see that recognition of the '3' has decreased, misrecognition of the 'M' as a 'J' has increased significantly, and other areas of misrecognition have become more prevalent. We are left to assume that the neural net has begun overfitting at this point.

One piece of the story that these diagrams do not tell is that a few of the apparently missed letters were actually guessed correctly, but with very little confidence. While it is easy to assume that some letters like 'X' train well because they are somewhat distinctive from the other letters, it is harder to say why letters like 'S' and 'R' are barely recognized at all. Moreover, repeated trials with the same parameters lead to the same results; the character '5' tends to be recognized well, and 'S' does not. One explanation of this is that each of the neural nets started in the same place and thus found their way to the same minima; this theory will be explored later. Another interesting note is that the 'M' character tended to be recognized as a 'J' with varying degrees of certainty through all of the experiments. This phenomenon is also currently lacking explanation.

2) Testing Nets on the Segmented Results

At this point, it seemed reasonable to test the nets against the output of the character segmentation. Note that at this point we had not attempted to deal in any way with the rotation or skew of the characters being passed in.

Applying the better of the previously discussed neural networks to a test set of segmented characters, we received our first indication of how the net will perform the actual task for which we have designed it. Unfortunately, this test did not perform up to hopes. Here, we are less concerned with how strongly the net recognizes the character as each letter or number, but rather which of the letters and numbers it thinks is most likely.

The resulting identification produced a success rate of approximately 17%. Unfortunately, this is far too low to give a 1% success rate on the CAPTCHAs. A 17% character identification rate yields a probability of getting 8 correct characters of $6.9\text{x}10^{-7}$.

The results do show some interesting things about the problem, however. Some images that one might expect the net to recognize easily, it did. The following characters were recognized strongly as an 'X' and a 'T', respectively.

However, other characters that appeared just as easy to decipher were recognized dramatically incorrectly.

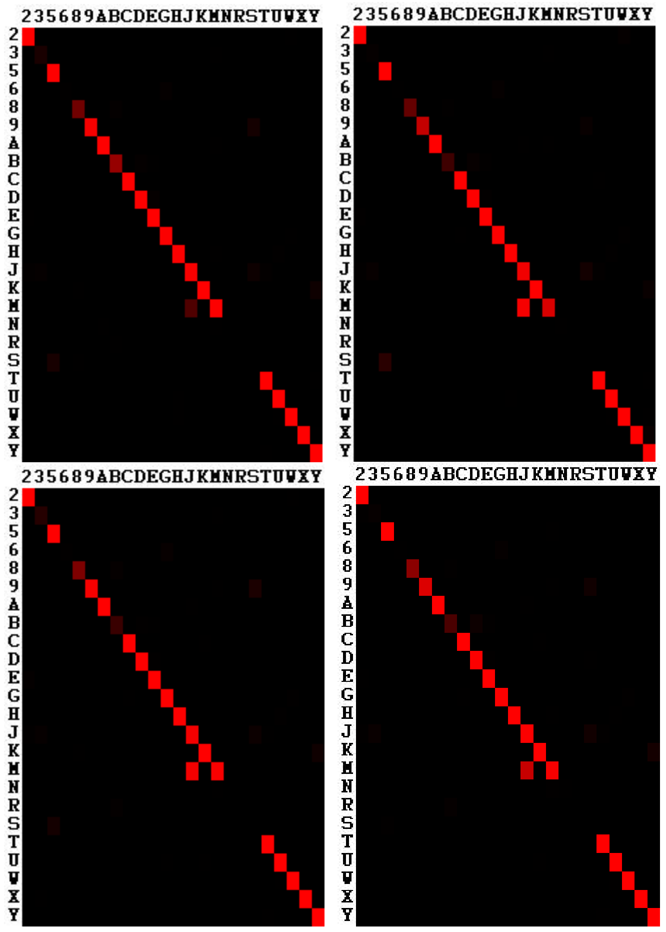
Both of the above characters were recognized strongly as '2'. One interesting thing about this observation is that X, T, A, and 5, were all characters that performed very well in validation against an unknown font, and these samples do not appear to differ dramatically from the actual fonts.

At this point, however, we were hopeful that the large discrepancy between the recognition rate on the unknown font and the segmented character images could be accounted for by dealing appropriately with rotation and skew.

D. Randomizing and Restarting

Before rotation and skew were dealt with, it seemed worthwhile to assure that the neural networks were not simply getting stuck in local minima that prevented them from living up to their full character recognition potential. In order to account for this possibility, we designed an experiment whereby four copies of the neural network would start with four randomized sets of weights, and run in parallel. For this experiment, we used a learning rate of 0.06, a momentum of 0.03, and a hidden layer size of 300 nodes. Each of the four nets trained for 1500 cycles.

When these networks completed training, they were compared with one another. All of the final training RMSE values were 0.08391±0.00001. It quickly became clear that the randomization of the starting point was not going to dramatically affect the results. As another point of comparison, consider the tests of each of the four on the validation 13th font:



While each of these four networks does have some subtle variation (easy points of comparison are the misidentification of M as J, and the amount that S was identified as a 5), they present the exact same patterns of error as one another. This led us to believe that merely restarting the training at different random points was not going to lead to a dramatically better solution.

E. Dealing with Character Rotation and Distortion

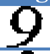
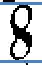
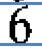
Our greatest hope, then, for lifting the character recognition to an acceptable level lied within accounting for the rotation of the images, as this seems to be the most dramatic (and seemingly important) difference between the unknown font test set and the segmented images from the CAPTCHA. We considered to primary plans of attack for this problem. The first plan would deal with the rotation external to the neural network, and the second would force the neural net to do the work for us. We will address each individually.

1) Dealing with Rotation Externally

The basic idea of this approach was straightforward. Instead of attempting to recognize just the original image, we would use the neural network to attempt to recognize three different images. The first image would be the original, the second would be the original rotated 15 degrees to the left, and the third would be the original rotated 15 degrees to the right. Since there is a strength associated with the recognition of the characters, we would then simply take the result of whichever of the three was identified most strongly.

Unfortunately, this idea had to be rejected before it was ever even experimented with. The problem with the scheme was found when looking at the results from the un-rotated network on the segmented images. The inherent assumption in this approach is that the characters that are recognized incorrectly are recognized significantly less strongly than those that are recognized correctly. Looking at the previous results, we can see that this assumption is patently false.

Consider the following set of misidentifications:

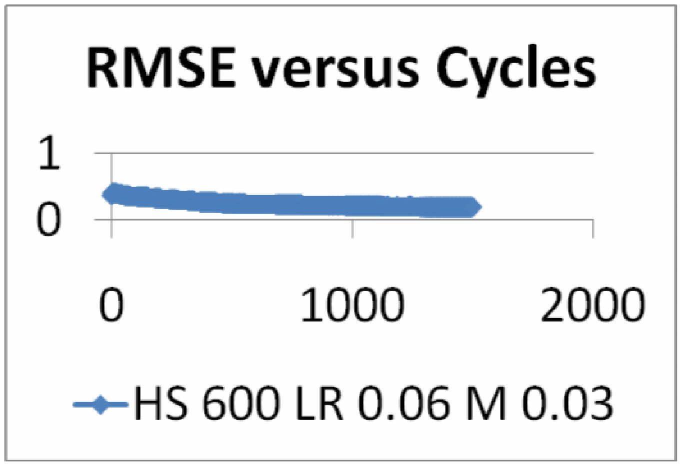
Image	Identified As	Strength
	U	0.9999764736993
	H	0.9976125625314
	M	0.997241982241

Not only is it disappointing that these numbers which appear only marginally distorted are misidentified, but each is misidentified with strengths approaching one. This quickly showed that identifying the image three different ways would not lead us to a useful solution.

2) Dealing with the Rotation within the Neural Network

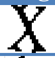
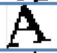
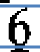
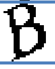
The second possibility for dealing with the rotated images is training the neural network with them. This is in some sense the mirror image of the previous strategy. Here, we multiply the training set by three. Each image is tripled by adding an image of it rotated 15 degrees left and another of it rotated 15 degrees right. Then, instead of 24 output nodes, the net is given 72. Thus, it recognizes the different rotations of the characters differently.

The first challenge with this configuration was training time. Whereas in the starting weight randomization experiment all four nets were trained for 1500 cycles in about 16 hours, this network took over 24 hours for 1500 cycles by itself. This of course can be explained by the much larger sample size and added complexity of the output nodes, however it did prevent us from doing more testing on different approaches to the rotation. The following chart shows the changing RMSE through the cycles:



The chart shows just how slowly the learning was taking place for this neural network. After 1500 cycles, the RMSE on the training set was only at approximately 0.19. Nonetheless, we wanted to forge onward and test this neural network on the actual data sets.

When we applied this neural network to the actual segmented images, it seemed we were again thwarted. This time, the results were actually significantly worse than the neural networks that did not attempt to account for rotation. In fact, it recognized about 95% of the images as '2's, generally very strongly. Some examples of the misidentifications:

Image	Identified As	Strength
	2	0.9999999414222
	2	0.9920851667093
	2	0.9999999999544
	2	0.9999999996914

As such, the long training time of this particular network failed to produce better results for the task of recognizing the rotated and skewed characters.

VII. Conclusions

The image-processing phase of solving this CAPTCHA supports the conclusions of [13] that the mechanisms employed in this CAPTCHA to stymie segmentation efforts are currently effective. The perturbations that contributed most significantly to the challenge of segmenting these images were a combination of the widely varying glyph sizes, glyph-intersecting noise arcs and effectively varying stroke widths in the glyphs.

If the non-noise glyphs in this CAPTCHA been of uniform size, segmentation would have been reducible to selecting glyph-sized areas of the image containing the largest-number of connected points. Computation time for such an approach would be minimal, and results could be expected to be quite satisfactory.

If the non-noise glyphs had been of varying, but similar sizes, segmentation would have been more difficult than the constant-size case, but still relatively easy. In that case, starting from the center of the image and working out until the narrowest point is found could be expected to be highly accurate.

Without glyph-intersecting noise arcs, segmentation of this CAPTCHA would have immediately been reduced to finding sets of connected points, which is an easily solved problem. The only remaining issue at that point would be to discard noise arcs as best as possible to reduce the number of sub-images for the neural network to consider.

The design of this CAPTCHA uses a localized warping [13] to give the appearance of varying stroke widths in the glyphs and noise arcs. It is likely that, without this distortion, removal of noise arcs based on stroke width would have been possible. The warping used to achieve this effect is the same as that used to distort the characters, except with a higher cut-off for the low pass filter and a lower intensity to yield a more localized distortion.

Attempting to tune the parameters of our neural networks taught us several significant lessons about neural network design. First, for complex problems such as character recognition, keeping the learning rate low is critical. Although this reduces the speed at which the network converges significantly, it allows the network to eventually reach better RMSE's. Momentum should also be kept at a reasonably low value. We also learned that input image resolution was best kept at a reasonable middle value, as to give enough information to the neural network without providing too much noise. Finally, we learned that increasing the hidden size too dramatically would lead to noticeable over-fitting.

Our neural networks turned out to be reasonably good at OCR when the characters in question were not distorted. It seems that the technique of generating images in a multitude of fonts as a training set is a viable one. This may have worked better if the character images were normalized in some way. Nonetheless, the better of the networks that we trained produced an accuracy rate of approximately 92% on undistorted characters.

The results of the neural networks attempts to recognize the characters that came from the CAPTCHAs, however, were not nearly as positive. Our best neural networks for the task recognized only 17% of the characters correctly, with the resulting probability of getting eight characters in a row correct being abysmally low. This seems to have to do primarily with the distortion of the letters, both in terms of rotation and skew. Even discounting the characters that were not segmented well enough for the neural network to have much of a chance of recognition, the percentages were still quite bad.

As far as the neural network character recognition is concerned, though our results were not particularly positive on the CAPTCHAs characters, we still believe the general approach is viable. The successes of other neural networks on similar tasks in the past leads us to believe that with different training parameters and training sets, we may be able to get significantly better results.

It is clear that the combination of the challenging segmentation and the distortion of the letters involved will make this particular CAPTCHA challenging to break for some time to come.

VIII. Future Work

As the success rate of breaking CAPTCHAs is dependent upon the image processing and segmentation phase and character recognition phases, it stands to reason that improvements in one or both areas should provide an improvement in success rates. As a result, we recommend that future work target both portions for best results.

A. Image Processing

There are a number of image processing techniques that could have been applied to the image that hold promise for improving character recognition efficiency that were not tested due to time constraints.

One image processing technique that holds promise for improving character recognition efficacy is "thinning", which results in a skeletal representation of the characters, preventing the variable stroke width from influencing the neural network's learning and recognition. If the CAPTCHA used a fixed font, or consistent stroke widths this approach would likely reduce the network's efficiency, however in the presence of inconsistent stroke widths, it would remove a noise factor from consideration.

A parallel thinning algorithm for binarized (monochrome) images is described in detail by Zhang and Suen in [14] . In brief, the algorithm determines whether a pixel should be retained based on the eight neighboring pixels. There are four conditions which must be met by a pixel to be retained in the algorithm in [14], which are:

- The pixel must have between 2 and 6 inclusive non-zero neighbors
- Starting from the neighbor directly north of the pixel and continuing clockwise, there must be exactly one pair of sequential neighbors whose values form a 01 sequence
- At least one of the pixels directly above, to the right of, or below the pixel must be 0
- At least one of the pixels directly to the right of, below or to the left of the pixel must be 0

Preliminary results with this approach show promise by removing irrelevant features from the glyph. Figures VIII-1 and VIII-2 show a glyph before and after application of the thinning algorithm.

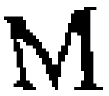


Figure VIII-1: Glyph Prior to Thinning Algorithm

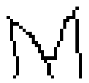


Figure VIII-2: Glyph After Thinning Algorithm

Recognition performance may be further improved by checking for and removing skew if it is detected. One approach to detecting skew is to find the left-most and right-most positions with the highest and lowest non-zero pixels, and calculate slope based on that information. To find these values, move in from the edges as long as the top is increasing or the bottom is decreasing, stopping for a corner when the top decreases or bottom increases. This will result in stopping at local minima and maxima, but should yield the desired result in most cases. Figure VIII-3 below shows a glyph whose corners, as detected by this algorithm, are marked with green circles on the graph of heights shown in Figure VIII-4.

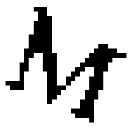


Figure VIII-3: Skewed Image

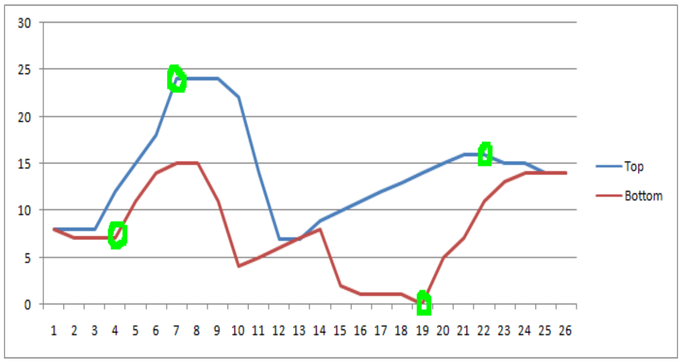


Figure VIII-4: Marked Corners for Figure VIII-3

Once the corners have been determined, the slope from the lower-left to upper-left and lower-right to upper-right corners can be determined. If the two slopes are the same, or very nearly the same, then the image can be de-skewed by rotating the image such that the slope on the left and right sides are vertical. The circumstances under which this technique would most likely fail, including the letter *L* are of less concern for this CAPTCHA due to problematic letters not being part of the set of characters encountered.

The segmentation portion of image processing could be improved by calculating the weights of the columns of the image and generating a tree of the weights, rooted at the center of the image with a branching factor of one. Performing an iterative deepening search of this tree should yield the narrowest point nearest the center of the image. This approach would be preferable to naively splitting the image down the center due to the possibility that the image is comprised of two glyphs of significantly different size, which could cause a character to be split internally. One problem with this approach is that it would not effectively split segments where three glyphs are connected by a noise arc, however this is somewhat mitigated by the fact that the segment splitting is recursive.

Character recognition is also adversely affected by the presence of the noise arcs. It is possible to find likely noise arcs in glyphs, but there is a non-trivial risk of discarding portions of characters during this process. To minimize this risk, an attempt should be made to recognize the glyph prior to attempting to erase noise arcs. If the glyph cannot be recognized, then noise can be attacked from the left and right-most edges using the column weights. From the outermost columns, work inwards replacing the column with background color until the first spike in weight occurs. The minimum weight that can be considered a spike must be determined through experimentation, but would likely be in the neighborhood of one or two pixels. In the presence of letters such as *L*, this technique would not be appropriate, as the leg of the L would be chipped away, leaving an apparent *I*. Fortunately, this CAPTCHA does not feature the character *L*, and thus this approach remains potentially applicable.

B. Character Recognition

Future work for the neural network character recognition could take many different forms. One primary thrust of this work could be to apply a more complex neural network to a much larger data set taken directly from the CAPTCHA problem. It is likely that this approach would require a very long learning time, but it may generalize better than the pristinely rendered fonts used in our testing.

Another possibility would be to use the same training data that was used in our experiments on dealing with the rotation, but to allow the neural network to classify the set as simply the 24 characters instead of the 72 characters + rotated characters. This approach has been used by other neural networks in the past. Again, it is expected that the amount of time taken to train the neural network would go up significantly.

One possible problem that was encountered in the neural net experiment was that of normalization. It was not clear how much of the error was being caused by the inherent differences in the way that characters were presented in the training set versus the characters from the CAPTCHAs. While the training set images were rendered by placing a 72 point font in the middle of an 80x80 image, and the actual sets images were form-fitted around the character. We attempted to correct for this by drawing the CAPTCHA images into a larger square image, thus making them look like the original ones, but it is impossible to tell if this corrected the problem appropriately. In the future, it would be interesting to repeat the experiment with a better strategy of this type of normalization.

In actuality, a longer time window and more significant hardware would allow the running of a much larger number of experiments, and thus give us a better chance of finding a solution that works. Since every change of the methodology required several days of training of the networks, we were limited in the amount of iterations that could be performed.

In a completely different vein, the character recognition portion of the experiment could stop using neural networks entirely and make use of a different classification construct, such as Support Vector Machines (SVM), instead. These have been used successfully in the past for this purpose as well.

IX. References

[1] "Yahoo! Mail," 2007.

[2] "Gmail," 2007.

[3] "Ticketmaster," 2007.

[4] K. Chellapilla, K. Larson, P. Simard, and M. Czerwinski, "Designing Human Friendly Human Interaction Proofs (HIPs)," in *SIGCHI conference on Human factors in computing systems*, Portland, Oregon, USA., 2005.

[5] "MSN Hotmail," 2007.

[6] G. Mori and J. Malik, "Recognizing objects in adversarial clutter: breaking a visual CAPTCHA," 2003, pp. I-134-I-141 vol.1.

[7] K. Chellapilla and P. Simard, "Using Machine Learning to Break Visual Human Interaction Proofs (HIPs)." vol. 17, 2004.

[8] A. Rajavelu, M. T. Musavi, and M. V. Shirvaikar, "A neural network approach to character recognition." vol. 2: Elsevier Science Ltd. Oxford, UK, UK, 1989, pp. 387-393.

[9] A. K. Jain, J. Mao, and K. M. Mohiuddin, "Artificial neural networks: a tutorial." vol. 29, 1996, pp. 31-44.

[10] I. Guyon, P. Albrecht, Y. Le Cun, J. Denker, and W. Hubbard, "Design of a neural network character recognizer for a touch terminal." vol. 24: Elsevier Science Inc. New York, NY, USA, 1990, pp. 105-119.

[11] R. Romero, D. S. Touretzky, and R. H. Thibadeau, "Optical Chinese character recognition using probabilistic neural networks." vol. 30: Elsevier Science, 1997, pp. 1279-1292.

[12] "Joone - Java Object Oriented Neural Engine," 2007.

[13] Simard, P. Y., R. Szeliski, et al. (2003). Using character recognition and segmentation to tell computer from humans.

[14] Zhang, T. Y. and C. Y. Suen (1984). "A fast parallel algorithm for thinning digital patterns." Commun. ACM **27**(3): 236-239.