# TJHSST Computer Systems Lab Senior Research Project: Breaking a Visual Captcha 2006-2007

Tianuhi Cai

May 23, 2008

## Abstract

CAPTCHAs, Completely Automated Public Turing tests to tell Computers and Humans Apart, are tests to determine if a user is human. They are often found on registration webpages to prevent automated signups and spam. The goal is that its challenges are easy for humans to solve, but difficult for computers to solve. Common variants are audio and visual CAPTCHAs, which consist of an image with letters or numbers that are to be typed in to a form by the user. The goal of this project is to devise a system to break a particular CAPTCHA using image processing, optical character recognition, and an artificial neural network.

**Keywords:** CAPTCHA, turing test, median filter, optical character recognition, OCR, neural networks, computer vision

## 1 Introduction

Completely Automated Public Turing tests to tell Computers and Humans Apart (CAPTCHAs) are tests to determine if a user is human. They are often found on registration pages, such as when registering for an email address at Gmail. It is sometimes referred to as a reverse turing test, since it is a test administered by a computer to distinguish whether the user is human or computer, rather than a test in which a human questions whether a user is human or computer. It is a system with challenges that are easy for humans to solve, but difficult for computers to solve.

Websites often use variants such as audio and visual CAPTCHAs. Visual CAPTCHAs consist of an image with letters or numbers that are to be typed in to a form by the user. Often, the letters are moved and rotated, noise is added, and the image is distorted, depending on the specific CAPTCHA. In this case, the problem may be solved by the computer, using computer vision techniques, as a combination of image processing and character recognition. The image is extracted from the web page, background clutter (such as noise) is removed, segmentation (separating letters) is performed, and the letters are identified.

The goal of this project is to break a visual CAPTCHA - specifically the one at captchas.net, which is provided for free. It is a black and white image with 6-16 letters (depending on specifications) that are all lowercase, rotated, and translated. Black and white noise is also added.

There are two parts in the project: getting rid of noise as well as other image processing (segmentation), and the use of the artificial neural network.

### 1.1 Background

A CAPTCHA's purpose is to distinguish between a human and a computer by presenting a challenge that is easy for most humans, but difficult or impossible for a computer. Visual CAPTCHAs present a user with an image with letters or numbers, and the user is to enter the letters or numbers into a form. The image is often constructed such that the letters or numbers are distorted, or contain noisy data. However, visual CAPTCHAs often do not stand up to this challenge. Many have been broken in the past, and this research project will attempt to do something similar by using image processing and character recognition, along with a neural network. Alternatives for the more easily broken visual captchas are ones that include logic (such as a captcha with a math problem), logic questions without an image (though it would take a large database of questions and answers), a classification problem (showing a user a variety of pictures depicting the same thing, and expecting an answer), or an auditory CAPTCHA (in which the user listens to a file and transcribes the words).

The goal of this project is to break the captcha at captchas.net, which provides a free

CAPTCHA service. The CAPTCHAs they provide are in black and white, with six or more (user-defined) lowercase letters of the same font. They are rotated and translated, and the image has a significant amount of black and white noise.

The noise will be removed using a variation on a median filter. The letters will be separated using a flood-fill. The letters will be identified using an artificial neural network.

If there is time, CAPTCHAs from sources outside of captchas.net will be used.

The artificial neural network will be used to match the black and white image of a letter to the letter itself. Neural networks are important in that they can model highly complex, nonlinear systems and can be proficient in classification and pattern recognition.

Before using the neural network to classify images, it must first be trained. In this case, inputs of image and letter pairs will be used to train the network. The actual letters of the CAPTCHA can be identified using a formula provided by captchas.net, as the letters in the image correlate to a function of the address.

Training, in this case, will be done through backpropagation. Backpropagation is a standard neural network supervised training algorithm.

In an artificial neural network, a group of simple neurons are used to display a more complex behavior as a group, which is determined by their connections and parameters. There is generally an input layer, an output layer, and possibly many hidden layers. The input layer takes in data, the output layer spits out data, and hidden layers do intermediate processing. Neurons fire with a value between 0 and 1; when a neuron receives input, it weights the inputs by neurons connected to it, and determines whether or not to fire by the sum of the weighted inputs. A backpropagation network, in short, starts out with a set of randomized weights, and adjusts the weights on each layer depending on the error produced. In this project, the neural networks will use supervised learning. This is where the neural network is trained with a set of example pairs, and where the goal is to find a set of weights that can match the pairs.

Research on neural networks has been in existence for several decades. In particular, the use of neural networks for classification has been used.

Although neural networks do not accurately represent their biological counterparts, they have been shown to successfully classify images, as long as a large enough training sample is available.

Greg Mori and Jitendra Malik, from the UC Berkeley Computer Vision Group, broke the Gimpy and EZ-Gimpy CAPTCHAs. Their research is documented in their paper, Recognizing Objects in Adversarial Clutter: Breaking a Visual CAPTCHA, as well as in their website at http://www.cs.sfu.ca/ mori/research/gimpy/. The challenges they faced were much harder than the ones in my project, as EZ-Gimpy included severe distortions, varied fonts, and a really noisy background. In addition, GIMPY contained multiple pairs of words overlapping each other, in the same color. Because their challenge is different, the techniques they use are not all applicable to my project. For instance, I do not have to deal with overlapping characters, nor do I identify entire words. However, much of the beginning process is similar: remove noise, separate out letters, and identify letters.

Handwritten Digit Recognition with a Back-Propagation Network by Y. Le Cun et al at AT and T Bell Laboratories used a large back-propagation network to read hand-written zip codes. It took inputs in the form of black and white images. The project focused on the effect of architecture on the ability of the network to recognize digits. Its architecture was modeled in such a way that not all neurons from each layer were connected; there were five layers in total, with some neurons receiving only local input. This architecture performed well in identifying the handwritten digits, regardless of position.

## 1.2 Expected results

A backpropagation neural network was written in Java, for the classification of letters. Image processing programs were also written to reduce background noise and extract letters from the CAPTCHA image. Research, design, programming, and testing will all be done throughout the year as modifications are made to the program. The testing and training images are downloaded from the internet, from captchas.net, using the ruby programming language. For the language, JAVA will be used, and JGrasp would be used for software.

The goal of this project is to develop a program that will break visual CAPTCHAs, such as the one at captchas.net. The goal is also to learn more about different types of image processing and the steps in OCR, as well as how neural networks work. It is also to gather evidence of how this is best done, and to show weaknesses of basic visual CAPTCHAs and to provide support for other types of captchas such as logical ones and math-based ones.

### 1.3 Type of research

The type of research in this project is use-inspired basic research, to pursue fundamental understanding but motivated by a question of use (Pasteur's work on biologic bases of fermentation and disease)

## 2 Procedures and Methodology

The project will be deemed successful if the neural networks created can successfully identify the letters presented in a CAPTCHA.

To test if the program works, a neural network will be trained with test data. Then, the network will be tested with data that was not from the training sample. If the program can accurately identify the characters, it will be deemed successful.

Because there are multiple parts of the project that focus on different areas, each of these parts may be tested separately. The neural network was tested earlier in the year using simple binary functions such as AND, OR, and XOR, and was found to perform well. The image processing program was also tested earlier this year to see if it could reduce the high amount of noise in the CAPTCHA images using a median filter. It worked fairly well. The part of the program that retrieved images from the internet also worked well; it downloaded fifty images rather quickly from captchas.net and named the images according to the text shown in the captcha using a particular formula provided by captchas.net. The next step is to connect the pieces and test the whole thing at once.

## 3 Development

Two large sections - neural networks and image processing - have been worked on in the past three quarters. In addition, the downloading of the images was finished.

### 3.1 Neural Network

In the first quarter, a working back-propagation neural network class was created in Java. A neuron layer class was used, and a neuron class was used for the neuron layer class.

The final back-propagation neural network class was written with three layers - one input, one output, and one hidden. Each node in each layer is connected to each node in the next layer; the number of nodes in each layer varies depending on constructor input.

Neurons from the input layer would receive input, and pass forward a signal to the neurons in the hidden layer, which would pass a signal to those in the output layer. The output of each neuron depends on the output of the neurons that feed into it, as well as the weights that connect it to the neurons that feed into it.

Then, once an output is obtained, it is compared to the correct output. The network then uses back-propagation to adjust the weights in the layers in order to obtain a more correct solution the next time around. The speed at which the weights conform to the input is dependent on the learning rate, which is also specified.

The neural network was tested initially for binary operations, such as AND, OR, and XOR.

Later on, the network was adjusted to learn rudimentary numbers. The inputs, as always, were numbers – each pixel in the number image was a 0 or a 1, representing white and black, respectively. These "images" were actually text files with these numerical representations.

The neural network succeeded in both instances. Although the testing was not a lot, it did demonstrate that it worked.

The weights of the neural network are able to be saved and re-loaded so that it can be trained multiple times without starting from random weights with the help of a saving and loading function I wrote.

### 3.2 Image Processing

During the second quarter, the image processing side of the problem was worked on. Java, again, was used, despite the fact that C or C++ may be better for dealing with images. However, it would be cleaner to use one language for the whole project, even if the output of the image processing is a text file of numbers (regardless of language). Luckily, Java's ImageIO classes and BufferedImage classes helped facilitate this.

The first step was to retrieve the images. ImageIO has a command for retrieving an image from the internet, bypassing wget and other terminal commands to download images from the internet. In addition, ImageIO allows loading a file into a BufferedImage, which is convenient.

After that, I wrote a method to convert the BufferedImage into an int[][] array, which would be easier to work with, as well as a method to convert the int[][] array back into a BufferedImage (to save).

I then wrote filters for the image, such as a black and white filter, which takes a threshold and makes everything under that threshold one color, and everything other that threshold another color. In addition, I wrote a modified median filter. Rather than taking the median of a large area, it takes the median of the 3x3 square surrounding a pixel, since noise is only on the scale of pixels. It is not a median filter in another way: it has the options to take the average of the middle three, five, seven, or all nine neighbors, rather than just the straight median. It was also written to be able to pick not only the median value, but also the least, greatest, second-greatest, and so on - as specified by an argument in the method.

After the images are processed, the int arrays that contain each letter are cropped and centered to fit the same size because a neural network must take the same number of inputs at each training / testing.

### 3.3 Image retrieval

The images were downloaded from captchas.net using a ruby script that automated downloading and saving. The images were named after the letters depicted in the image using a formula provided by captchas.net, which is crucial to the training of the neural network.

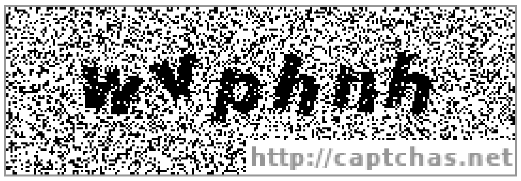

Figure 1: Original sample captcha

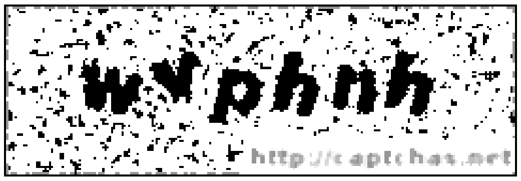

Figure 2: Median filter for 3x3 square

This was then tested on images retrieved from captchas.net.

Surprisingly, the extra features written into the median filter were useful. I chose to use a median filter rather than a gaussian blur for two reasons: a gaussian blur requires more coding (because of the statistical aspect), and a gaussian blur makes a shape lose the definition in the edges.

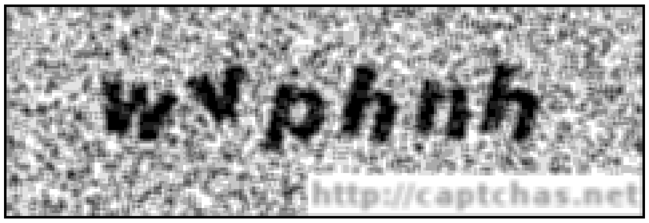

Figure 3: Averaging 3x3 square

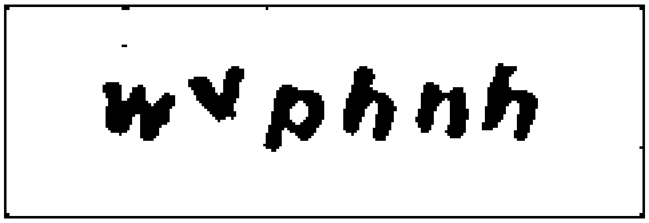

Figure 4: Captcha, after a few rounds of median-filtering. The shape is less sharp, but noise is decreased.

The fact that not the exact median can be chosen also became useful to remove mostly dark-colored noise, because there was a lot of it, and it helped to lighten the picture. A slight blurring was combined with the modified median filter, and then it was changed to black and white to remove grey areas.

The code I wrote also flood-fills letters. This means that letters must be continuous and not touch other letters, limiting the function of this program. However, it still suits the samples at captchas.net.

## 4 Discussion, Conclusion, Recommendations

The purpose of this project was to develop a program that could break a CAPTCHA, thereby impersonating a human. Because this project is not yet complete, a discussion has not been finished. It is underway. Conclusions and recommendations have not been made yet.

Right now, possible future changes include segmenting letters that are stuck together by splitting the image with a vertical line, because the current segmentation algorithm using flood-fill will not be able to accommodate this.

## 5 Appendices

### 5.1 Code - neural network

Code contains neural net class, neural net layer class, and neuron class. However, this is not included because if all the code was actually here, the report would be 34 pages.

### 5.2 Code - image processing

Code contains image processing methods, as well as a sample implementation. Methods not shown – it takes too much paper, and I'm turning in the code separately.

## 6 Literature Cited

## References

[1] Le Cun et al, "Handwritten Digit Recognition with a Back-Propagation Network", AT and T Bell Laboratories, 1990.

[2] Hart, Anna, "Using Neural Networks for Classification Tasks - Some Experiments on Datasets and Practical Advice", Department of Mathematics and Statistics, Lancashire Polytechnic, 1992.

[3] Mori, Greg and Malik, Jitendra, "Recognizing Objects in Adversarial Clutter: Breaking a Visual CAPTCHA", UC Berkeley Computer Vision Group.

[4] Belongie et al, "Matching Shapes", Department of Electrical Engineering and Computer Science at UC Berkeley, 2001.

[5] Stuart Russell and Peter Norvig, Artificial Intelligence - A Modern Approach, Prentice Hall, 2003.

## 7 Acknowledgements

I had a lot of help from Mr. Latimer, who provided knowledge and guidance.