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

Tianuhi Cai

June 10, 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, Java, Ruby

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. 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 six to sixteen 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: processing the images to extract letter images, and the use of the artificial neural network to identify those letters.

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 a visual CAPTCHA - specifically the one at captchas.net, which is provided for free. Their CAPTCHAs are black and white images with six to sixteen letters (depending on specifications) that are all lowercase, rotated, and translated. Black and white noise is also added.

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

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.

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.

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.

Research on neural networks has been in existence for several decades. In particular, the use of neural networks for classification has been used. Sherin M. Youssef and Shaza B. AbdelRahman researched license plate recognition in their paper, A Smart Access Control Using An Efficient License Plate Location And Recognition Approach. In addition, 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. Another group, Le Cun et al. at AT & T Bell Laboratories, used a large back-propagation network to read handwritten zip codes. Their network had a special architecture in which 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. The testing and training images are downloaded from the internet, from captchas.net, using the Ruby programming language. For the other programs, Java is used, and JGrasp is used as an IDE.

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.

It is expected that the finished, trained neural network will successfully identify letters in CAPTCHAs from captchas.net to a reasonable rate of accuracy.

2 Procedures and Methodology

The general procedure consists of several steps. The first step is the acquisition of the image, which is done by downloading them from captchas.net. This website provides a free CAPTCHA service, with a formula using the URL to tell you what an image says. The images were downloaded and named with Ruby, with filenames being the sequence of letters depicted in the image.

The second step is to remove background clutter. In this particular case, the CAPTCHAs provided contain a lot of black and white noise, which can be removed with a median filter. In contrast to a Gaussian blur, a median filter does not blur the image, thus saving fine details of the image while removing noise. Java was used.

The next step is segmentation separating out the letters from the background. It is performed in this project using flood-fill. Although this method has limitations, it is adequate for this purpose. The letters were also scaled and centered in a 9x11 box. This step is also done in Java. The images of the letters are turned into arrays and outputted into a text file as lines of decimals from 0 to 1, representing shades from white to black.

The last step is the identification of the characters that have been segmented. This is done with a three-layer backpropagation neural network. The network is first trained using 1742 letter images, read from a text file, so that it learns how to identify those characters. Afterwards, it is tested on a new set of 600 different letter images, using what it has learned to identify those images. A key feature of the neural network is that it can be saved into a file and reloaded.

Testing was done to gauge the effects of learning rate and training iterations on accuracy of identification.

3 Development

This project is made up of two large sections: neural networks and image processing, which were worked on in the first three quarters. The last quarter was spent putting them together.

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.

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.

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

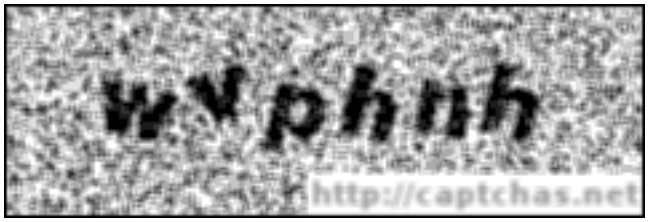

Figure 3: Averaging 3x3 square

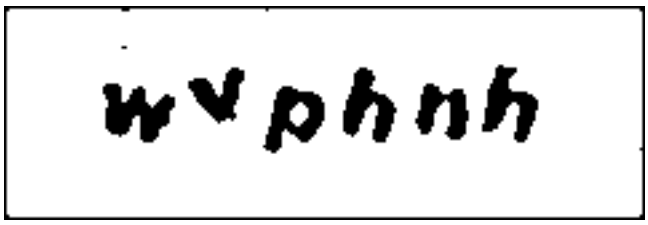

Figure 4: Captcha, after a few rounds of median-filtering. The shape is less sharp, but noise is decreased.

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.

3.4 Training and Testing

Before the neural network was trained, the image processing program was used to convert folders of CAPTCHA images into text files representing those images and the text inside them. Each letter was represented in two lines: the first line contained a shortened version of the image in terms of floating-point numbers from 0 to 1, with 0 being white and 1 being black. The second line said what letter it was.

Two Java programs were then written to train and test a neural network using specified text files for the neural network and training sets. The training program read in the inputs from the text files representing letters and used methods from the neural network program to train a specified neural network. It trained the neural network on 1742 different letter images. The testing program tested the specified neural network with 600 letter images different from the training set, displayed accuracy rates, as well as sets of letters that were wrongly identified. These were then run one after another.

During the training and testing stage, the variables of number of training iterations and learning rate were varied separately and analyzed.

4 Results

Initially, the program did not work at all. After training, during testing, it would be indecisive and often not pick any letter with a high degree of confidence. The inputs in the first try were 30 pixels by 30 pixels, or 900 input nodes. The neural network had a measly

50 hidden nodes and 26 output nodes. Later on, after rescaling all the letters to 9 pixels by 11 pixels, the neural network worked.

The following table shows accuracy rate of identifying one letter in which training iterations is varied and learning rate is kept constant at 0.1. A training iteration in this experiment consists of backpropagating a neural network with one letter/image pair. The neural network was tested on a set of 600 letter images, none of which were in the training data set. (Of course, accuracy rates are higher when test data is part of the training data.)

Number of Iterations	Accuracy
10000	53%
100000	87.2%
500000	91.5%
1000000	91.5%

The following table shows accuracy rate of identifying one letter in which learning rate is varied and training iterations is kept constant at 100000 training iterations.

Learning Rate	Accuracy
0.05	86%
0.1	87.2%
0.2	91.2%
0.4	87%

The trained neural networks often mistook certain letters for certain other letters. For example, it mixed up i's with l's, and often mistook w's for m's. In addition, p's and q's, a's and e's, h's and b's, u's and n's, s's and e's, and q's and g's were also mistaken for each other. This may be because of the blobbiness of the letters, making it harder to distinguish between them.

The maximum observed accuracy rate was 91.5%, which is decent and can break a captcha from captchas.net with a high chance of success. It was observed that the number of training iterations was directly related to accuracy up to a certain point, as rate of accuracy did not improve from 500000 training iterations to 1000000 iterations. This may be because it has reached the upper limit considering other variables. It was also observed that increased learning rate helps accuracy rate up to a point after which it decreases the accuracy. This may be because a low learning rate has the neural network learn too slowly, whereas a high learning rate may cause the neural network to overshoot when it learns something.

5 Discussion, Conclusion, Recommendations

The purpose of this project was to develop a program that could break a CAPTCHA, thereby impersonating a human.

As of now, many future changes can be made to improve this CAPTCHA breaker, since this is a first version. These include a better method of segmentation; the current version of this program uses a flood-fill algorithm, which would identify two letters stuck together as one letter. A better way to do this would be to divide up the image with vertical lines wherever the least "ink" is. Another possibility is to eliminate noise better and reduce blobbyness, since the letter outputs are rather indistinct for certain letters. The neural network could also be improved by training it more on the letters that it has a hard time with, such as q's and g's. Another possible improvement is the usage of momentum, in

which the learning rate varies for better learning. The structure of the neural net may also be altered for better improvement, such as using pruning as in the experiment done by Le Cun et al. at AT & T Bell Laboratories. Another simpler possibility that was not explored because of time constraints is that of adjusting the number of hidden nodes.

Currently, the program only performs on a simple CAPTCHA - one without too much noise or distortion. In the future, these aspects would be improved upon; the program should also be able to deal with more noise, overlapped letters, and highly distorted letters.

One interesting possibility involves using a shape context, which was a method invented by Serge Belongie et al. and involves describing a shape as a histogram of angles and distances of points on an edge. It can bypass many distortions such as scaling and rotation.

6 Literature Cited

References

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

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

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

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

[6] Youssef, Sherin & AbdelRahman, Shaza, "A Smart Access Control Using An Efficient License Plate Location And Recognition Approach", Department of Computer Engineering at the Arab Academy for Science and Technology, Egypt, 2008.

7 Acknowledgements

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