Using Wavelets to break CAPTCHAs

Michael Ramsey, Simon Nash, Kiichiro Nomoto

May 9, 2015

1 Introduction

CAPTCHA is an acronym for "Completely Automated Public Turning test to tell Computers and Humans Apart"; it is a program designed to protect websites from bots by generating and grading challenge-response tests [1]. These tests are easy for humans to solve but difficult for computers, so CAPTCHAs distinguish between humans and automated programs that send spams or attack websites. This program was first invented around 2000, and the term CAPTCHA was coined by researchers at Carnegie Mellon University in 2003 [1,2]. The most widely used ones are text-based CAPTCHAs. For example, an image of awkward letters or digits often appears when we make a purchase or sign up for an email account with a secure Internet site. The process cannot be completed unless we recognize letters on the image and type the letters correctly. As computer performance increases, CAPTCHAs have also been improved in order to counteract this and to avoid being broken by computer programs. CAPTCHAs must be unreadable by computers, so letters of normal fonts we use daily on our computers cannot be used for CAPTCHAs. Text-based CAPTCHAs are usually distorted strings so that automated programs cannot read them. Twisted or tilted strings are most commonly used for CAPTCHAs. A grid and gradient background also make CAPTCHAs difficult to interpret by computers. An example of a CAPTCHA is present in Figure 1.

In addition to the text-based CAPTCHAs, there are also other types of CAPTCHAs such as image-based and video-based CAPTCHAs [3]. On a basic image-based CAPTCHA, abilities in image classification or image recognition are tested. Goswami, Powell, Vatsa Singh and Noore performed an experiment to measure vulnerability of text-based and image-based CAPTCHAs.

For this experiment, various types of CAPTCHAs were interpreted by machines that have abilities in recognition of text and geometric patterns. The results of this test concluded that the success rates for text-based CAPTCHAs were much higher than for image-based [3]. This means that the basic methods to break text-based CAPTCHAs are already established. Text-based CAPTCHAs might fail to distinguish between humans and computers, and cannot stop attacks from automated programs in this current age.

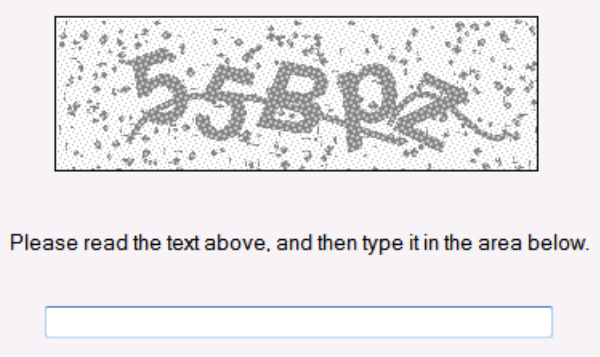


Figure 1: An example of a text based CAPTCHA.

This paper will report how we build a program to break text based CAPTCHAs on Matlab. In the next section, it will explain how to process CAPTCHAs so that computers can interpret them. We will mostly follow the method outlined in the paper "Breaking the Holiday Inn Priority Club CAPTCHA" [2]. For this project we especially focus on how to use the wavelet transforms to process CAPTCHAs. Then we will analyze the results and determine the accuracy of our program. This paper will be concluded by explaining our results and discussing possible research in the future.

2 Procedure

2.1 Generating a sample CAPTCHA

The CAPTCHA that used to appear on the Holiday Inn's Priority Club web-page is one of the simplest CAPTCHAs, and relatively easy to process. Therefore we chose to work with this type of CAPTCHA as the basis behind our project. We begin with generating a string similar to the one present in the Holiday Inn CAPTCHA. This string consists of five gray characters of a uniform font type and size. The characters are randomly chosen from

the set {I, V, X, C, D, L, M, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9}. This limitation is not necessary, but we can make further progress in a shorter time by using fewer characters. The five characters are placed on a white background with a black grid. The string is aligned, but not parallel to the horizontal grid-lines; the string is tilted at some angle between -10° to 10° with respect to the horizontal axis. These conditions make an ideal text-based CAPTCHA as shown in Figure 2, and this CAPTCHA is converted to an uncompressed image file so that MATLAB can import and analyze the image as a matrix.

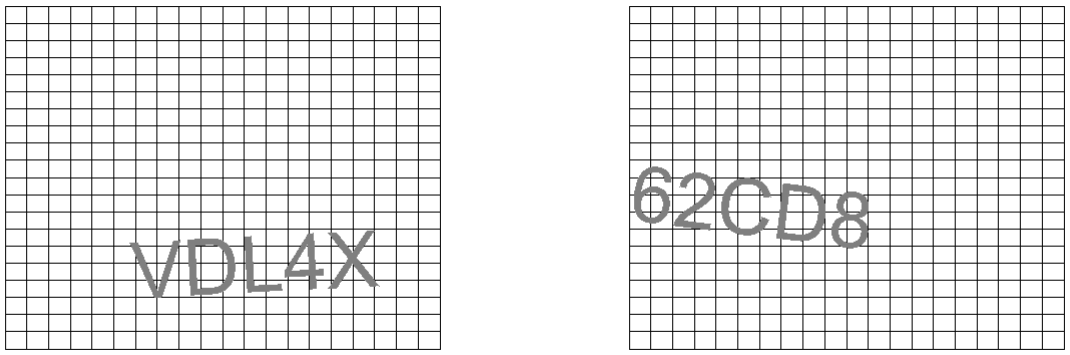
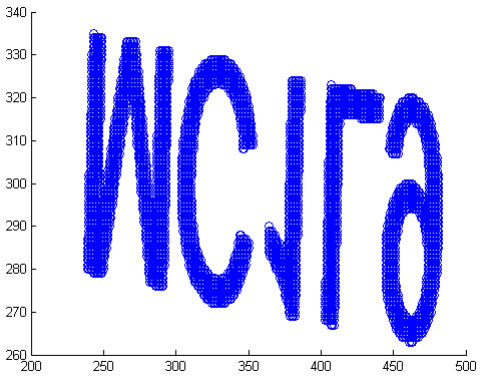


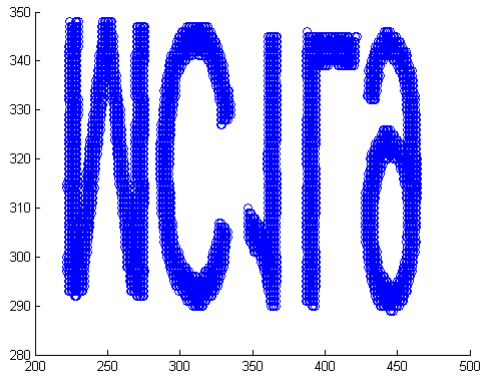
Figure 2: Two example CAPTCHAs generated by Matlab.

2.2 Isolating the characters

Several steps must be completed before we can apply the wavelet transformations to the CAPTCHA image. First we load our CAPTCHA into Matlab as a gray-scale image. The matrix of the image we created consists of only three different values of pixels. The pixel values equal to 0 correspond to the black grid-lines, the values of 255 store the white background, and the last is some value between 0 and 255, which represents the gray characters. For our matrix, the gray pixels had values of 127, so those pixels correspond to the CAPTCHA text. Then we isolated the gray pixels from the rest of the CAPTCHA and plotted the matrix coordinates (row and column number) on the x-y plane as scattered data points as shown in Figure 3. Since the gray pixels were plotted on the x-y coordinate system, we can apply a linear regression to find the best-fit line through those data points. The slope, m, of this line tells us how the string is tilted with respect to the horizontal axis. By calculating the angle $\theta = arctan(m)$, we can rotate each gray pixel by θ



(a) A scatter plot of our text pixels before rotation.



(b) The scatter plot of our text pixels after rotation.

Figure 3: A scatter plot and rotated scatter plot of CAPTCHA text.

around the origin so that the string is aligned horizontally. This is completed by applying the two-dimensional rotation matrix with our value of θ inserted. Since the point density of the string is not entirely symmetrical, we do not have a perfect rotation, however it is sufficient for character determination.

The next step is to put the rotated pixel locations back into a matrix. However this time we want the text to be white (a pixel value of 255) with a black background. Figure 4 contains an example of such a matrix. The black pixels have an important role for our program to isolate the characters of the CHAPTCHA text. We make a loop that detects columns of all zeros. If the zero - columns appear in series, those set of columns are defined as a space that separates two characters. For the last step of isolating the characters, each character is fitted into a matrix of size $2^n \times 2^n$. We do this because the Haar Wavelet transform requires a matrix size as a power of two so that we can perform multiple iterations of the transform. For each letter of CAPTCHA text, we obtain a matrix of size 64×64 by adding or deleting some rows and columns of all black pixels to the bottom and the right of the matrix. This puts the character at the top-left of the matrix which is important for character distinction and recognition.

2.3 Wavelet transforms for recognizing the characters

For this project, we use the Haar wavelet transform to process CAPTCHAs instead of the bivariate wavelet filters, which is used on the paper we refer to. The general role of the two dimensional wavelet transformation is to compress

Figure 4: The result after we place our rotated pixels back into a matrix.

an image file. Most of the information of an image can be concentrated in the blur portion, which is the upper-left quarter of the matrix after the transform. An example is present in the matrix in Equation(1) below where M is the original matrix, W is the wavelet transformation matrix and B is the blur portion of the transform. The numerical data of a large image file can be concentrated into the blur portion of size $2^{n/2} \times 2^{n/2}$, which is 1/4 the size of the original matrix. By iterating the wavelet transformation, we can improve this concentration even further. We originally had matrices of size 64×64 for each isolated character. Since they are are gray-scale images, we directly apply the transformations without any color mapping. By using 3 iterations, we concentrate the majority of the character information in the blur portion of size 8×8.

$$\mathbf{WMW^T} = \left[\begin{array}{c|c} \mathbf{B} & \mathbf{V} \\ \hline \mathbf{H} & \mathbf{D} \end{array}\right] \tag{1}$$

As we process CAPTCHA characters by the Haar wavelet transformation, we also create matrices for the characters I, V, X, C, D, L, M, 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9. We generate each character by using the same method as we used to make the sample CAPTCHA, but without rotating. These characters are of size 64×64 and are processed by the Haar wavelet transformation with 3 iterations. We result with 8×8 matrices; an example is shown in Figure 5(b). We store these matrices as row vectors in a CSV file which we call *character bank*.

Our program compares the isolated characters of the CAPTCHA text to every character in the character bank by calculating the matrix 2-norm between two matrices, which is defined as:

$$||A - B|| = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n}(a_{ij} - b_{ij})^2} \tag{2}$$

If two matrices are identical, the value of the norm is equal to zero. Since our sample CAPTCHAs are distorted, the characters isolated from our text and the corresponding character in the bank do not result with norms equal to zero. However, if our matrices have a similar arrangement of pixels, the norm becomes very small and is likely to be close to zero. In order to recognize the characters on our CAPTCHA image, our program calculates the matrix 2-norm with all of the character matrices in the character bank, and determines the matrix which minimizes the norm. Our CAPTCHA consists of five characters, so this process is repeated five times, and yields the most likely character. The CAPTCHA is considered to be broken by our system if the result matches all five of the characters in our CAPTCHA.

3 Results

We created 25 CAPTCHAs and ran them through our Matlab Program with a 100% success rate. For all 25 CAPTCHAs, our program was able to output the correct string of characters and the runtime for each CAPTCHA was less than 2 seconds. However this was not an immediate result and our algorithm required tinkering to get it performing correctly. Our first major problem occurred with the Haar Wavelet Transform. As said earlier, each character in our character bank is stored as a length 64 row vector. This row vector came from the 8×8 blur portion of our character matrix of size 64×64 after 3 iterations of the Haar Wavelet Transform. However we originally desired to perform four iterations of the Haar Wavelet Transform on the Character matrix, resulting in a blur portion of size 4×4. This would have been more efficient because there would have been less data storage required. If we had performed four iterations, now each character would have been stored as a vector of length 16, dividing the data storage by four as compared to four iterations. Four iterations of the transform would have been ideal, however when we ran our program, the character distinction was very poor. There did not even seem to be a pattern because the characters strings that were outputted did not even closely resemble the string in our CAPTCHA. Therefore in favor of effectiveness, we chose to reduce the number

(a) The result after isolating an M from a CAPTCHA string.

(b) The image we are comparing to.

Figure 5: An isolated character from a CAPTCHA string and the character that we are comparing it to.

of iterations to three. This still reduced the information needed to distinguish the characters from 64^2 to 8^2 pieces of data, a reduction of 98%.

Now our program was functioning at about 85% effectiveness at this point, but we desired a 100% success rate. Therefore we went back into our code and made one more big change. We chose to put the characters at the upper left of each character matrix by moving all rows of black to the bottom of the matrix. If you look back to figure 4, this was completed after this step. However we mentioned earlier that our linear regression was not perfect and leaves our string slanted occasionally. You can see this in figure 4. Therefore when we separated our characters, some of them still had rows of black at the top of the matrix. To fix this, we decided to first separate each character in their own individual matrix, then move all of the black rows to the bottom of the matrix. The result of this method is present in figure 5. This fix improved our character distinction to 100%.

4 Future research

Although we correctly identified all CAPTCHAs generated, there are many small improvements and investigations we did not have time for. Comparing the effectiveness of different wavelet transforms and an in depth investigation of the trade off between transform iterations and percent correctly identified characters would be natural ways to extend the project in the short term.

Having developed an adequate solution to the Holiday Inn CAPTCHA, it is worth considering how we could generalize our results to more modern systems, and the limitations we would face in doing so. The current state of the art is Google's reCAPTCHA system [4]. The system takes words which have

been scanned from books, but are incorrectly identified by Google's optical character recognition process. It then applies a series of random (sometimes noninvertable) transforms before presenting the user with a known control word, and unknown test word. If the user correctly identifies the control word, then the user's identification for the test word is used in future CAPTCHAs.

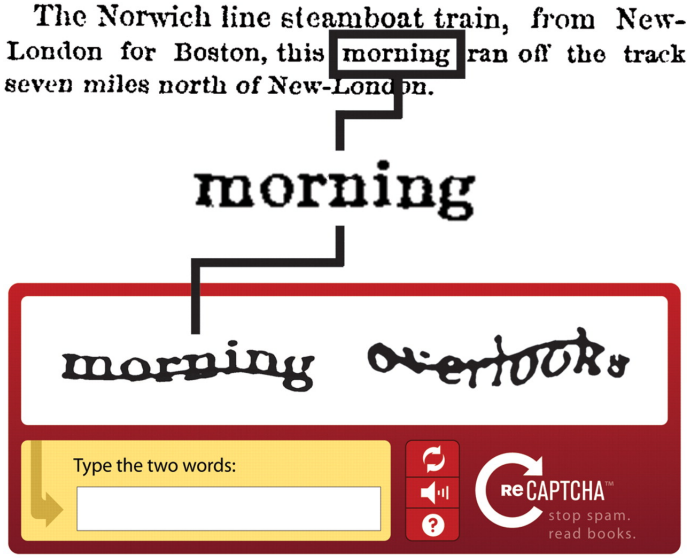


Figure 6: An example of reCAPTCHA.

The first modification we would complete are additions to our character bank, which currently only contains wavelet transformations for a subset of letters and digits in the Helvetica font. Since reCAPTCHA presents words from scanned documents, we would have no knowledge of what fonts are possible. Even considering a limit of 100 fonts, and considering only letters and digits, we would need to accept a character bank of length 3600. Since we need to compare the distance between each purposed character and each possible character, this would make the program run considerably slower.

The greatest trouble introduced in breaking reCAPTCHAs is the randomization of transforms. The ease with which we were able to de-rotate the text was entirely dependent on knowing exactly what kind of invertible transform they were performing. Once the CAPTCHA generation process is complicated by random, occasionally non-invertible transforms, applied in a random order, linear regression is no longer sufficient to invert these transformations.

However reCAPTCHA does have one weakness that the Holiday Inn CAPTCHA does not. Each CAPTCHA presented to the user is an English word, and therefore we could couple optical character recognition, with

English language models to produce reasonable guesses for strings where not every character could be confidently identified.

5 References

[1] The Official CAPTCHA Site, http://www.captcha.net/

[2] E. Aboufadel, J. Olsen, & J. Windle,
Breaking the Holiday Inn Priority Club CAPTCHA

[3] G. Goswami, B. M. Powell, M. Vatsa, R. Singh, & A. Noore,
FaceDCAPTCHA: Face detection based color image CAPTCHA

[4] L.Ahn, B. Maurer, C. McMillie, D. Abaham, & M. Blum
reCAPTCHA: Human-Based Character Recognition via Web Security Measures