Pattern Recognition ▮▮▮ (▮▮▮▮) ▮▮▮–▮▮▮

Contents lists available at ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/pr

Synthetic handwritten CAPTCHAs

Achint Oommen Thomas*, Amalia Rusu, Venu Govindaraju

CUBS, CEDAR, State University of New York at Buffalo, 201 Bell Hall, Amherst, NY 14280, USA

A R T I C L E I N F O

Article history:
Received 9 August 2008
Received in revised form 30 November 2008
Accepted 19 December 2008

Keywords:
HIP
CAPTCHA
Handwriting generation

A B S T R A C T

CAPTCHAs (completely automated public Turing test to tell computers and humans apart) are in common use today as a method for performing automated human verification online. The most popular type of CAPTCHA is the text recognition variety. However, many of the existing printed text CAPTCHAs have been broken by web-bots and are hence vulnerable to attack. We present an approach to use human-like handwriting for designing CAPTCHAs. A synthetic handwriting generation method is presented, where the generated textlines need to be as close as possible to human handwriting without being writer-specific. Such handwritten CAPTCHAs exploit the differential in handwriting reading proficiency between humans and machines. Test results show that when the generated textlines are further obfuscated with a set of deformations, machine recognition rates decrease considerably, compared to prior work, while human recognition rates remain the same.

1. Introduction

In 1950, Alan Turing described a test to distinguish humans from machines. This test, known as the Turing test [1], was designed to be administered by a human who would ask questions, to a human and a machine trying to pose as a human, and try to ascertain which is which by the answers received. In a reverse Turing test, a machine asks the questions with the aim of distinguishing between humans and machines. The CAPTCHA (completely automated public Turing test to tell computers and humans apart) [2] is an example of a reverse Turing test. Today, there are a number of online services which allow people to contribute content and interact online in some manner. Many of these services require that they be accessed only by humans. CAPTCHAs have come into the spotlight in cyber security applications for use in automated human verification online. Spam control for blogs and automated account sign-up by bots are some of the applications that require testing if the entity accessing a service is a human or an automated machine. There is economic incentive in posing as a human online. Consider a website like Ticketmaster.com which sells event tickets online. The website allows only a limited number of tickets to be bought using one account, to prevent activities like ticket hoarding which leads to ticket price inflation. A hacker could write a web-bot to sign up for hundreds of accounts and buy out a large number of tickets. These tickets could later be sold at a premium price, illegally. In such situations, it is imperative to know whether the entity creating the account is human or machine.

Most CAPTCHAs in use today are text-based. An image consisting of a series of printed text characters are rendered, distorted and obfuscated to varying degrees. The distorted image is then presented to a user. If the user correctly guesses the characters present in the CAPTCHA in the right order, he/she is granted access to some service. Circumventing the challenge posed by a CAPTCHA is an area that malicious hackers are actively looking into. Several printed text based CAPTCHAs have already been broken as reported in [3].

2. Background

Researchers are working on techniques to allow for automatically distinguishing between humans and machines. The general area is termed as human interactive proofs (HIPs) of which CAPTCHAs are a type of HIP. A number of different genres of CAPTCHAs exist; visual, auditory and semantic. The most commonly used genre is the visual CAPTCHA. Under the visual CAPTCHA, various types are present. http://www.captcha.net/cgi-bin/esp-pix [4] asks users to mark all images out of a set of images that contain similar objects. http://www.toallwhoseekit.net/cgi-bin/sq-pix [5] asks users to mark the region in an image containing some object where the object name is presented on screen to the user. However, the most popular kind of visual CAPTCHA is the text CAPTCHA. Text CAPTCHAs are popular since automatic recognition of degraded, noisy, distorted text with background clutter is still a challenging task for machines, but is a task that humans perform with relatively more ease. Many of the text CAPTCHAs proposed in the literature exploit this shortcoming.

* Corresponding author. Tel.: +1 716 472 6899; fax: +1 716 645 2377.
E-mail address: aothomas@buffalo.edu (A.O. Thomas).

0031-3203/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.patcog.2008.12.018

PessimalPrint [6], whose designers had extensive experience with OCR technology and understood its limitations takes advantage of known weaknesses of OCR technology. Other CAPTCHAs [7–9] are based on heuristics from Gestalt theory. ScatterType [7] uses the Gestalt principle of proximity. Fragments of the message are scattered within a short distance of each other; the relatively small distance allows the human perceptual system to connect the pieces so humans can read the message. Machines still find this to be a big challenge. For more background on CAPTCHAs, the reader can refer [10].

Various models of human-like writing generation are available in the literature [11–17]. Many of the existing approaches are on-line based since it is convenient to change the trajectory and shape of the letters based on the on-line information such as pen-down, pen-up, and velocity profiles. Lin and Wan [11] describe an approach to synthesize handwriting in a user's style. They collect handwritten character samples using a GUI interface and then build the textlines in a bottom-up fashion. Their technique adapts to a user's specific style. Wang et al. [12] presented a learning based approach to synthesizing cursive handwriting of a user by combining shape and physical models. A delta-log normal model based conditional sampling algorithm was used to produce the handwriting. Guerfali and Plamondon [13] describe the delta-log normal model which has been used for the generation and modelling of rapid movements to generate curvilinear strokes. An optimal selection of parameters for generation of these strokes will result in appropriate symbols being generated so as to conform to different characters in the alphabet. Kokula [14] performs script font ligature generation on-the-fly by optimizing a parametric curve between two characters. Researchers are also applying various character and image level perturbations directly on real character images in cases where online character information is absent. In [15,16], a perturbation model for generating synthetic textlines from existing cursively handwritten textlines by humans is presented. The perturbation model uses a continuous nonlinear function to geometrically transform points along the original textlines based on the value of the function. Mori et al. [17] developed a character generation method based on point correspondence between patterns. Their method automatically generates new character samples that appear to be written naturally.

3. Motivation

Automated recognition of high resolution printed text is all but a solved problem [18]. However, automated recognition of unconstrained handwriting continues to be a challenging research task. This fact can be exploited to develop human verification systems for cyber security applications. By replacing the printed text content in today's text CAPTCHAs with handwritten content, it would intuitively appear that the recognition task would be made more difficult for machines. To be suitable for online applications, a machine must be able to generate a challenge as well as score the response to it. Also, it must be possible to automatically generate infinitely many distinct artificially handwritten samples. To our knowledge, the only other use of handwritten CAPTCHAs is in the work done in [8,19]. However, in that work, the challenge images used were city names segmented out from postal envelopes. The approach of obtaining handwritten word images by segmenting out word images from handwritten documents (postal envelopes, handwritten letters, etc.) will seriously limit the size of the dataset from which challenges can be generated. It will also not include random strings of characters or combinations of phonemes. A finite dataset is a flaw while designing CAPTCHAs, as an adversary with access to the dataset can use dictionary/lexicon based brute force attacks to circumvent the system. The other approach would be to build textlines of handwritten words on-the-fly. This makes it possible to generate infinitely many distinct challenges limited only by the length of the textline. We have thus chosen to build a handwriting generator and base our generation technique on pre-existing character images taken from varied sources to reduce dependency on specific writer styles.

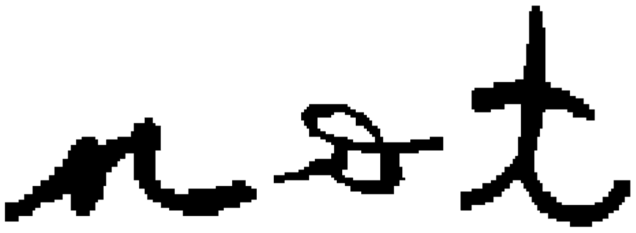
Fig. 1. Sample characters from the character dataset.

4. Generation method

In this section, we describe a method for the generation of cursive English handwritten textline samples that uses pre-existing character images. The generation algorithm consists of several steps: (i) character auto-scaling, (ii) automatic baseline determination, (iii) ligature endpoint detection, (iv) ligature parameterization, (v) ligature joining, (vi) skeleton perturbation, and (vii) skeleton thickening.

The high level algorithm to achieve this task can be described as follows. We first construct a preliminary image which is a concatenation of individual character templates/representations. Character templates are one pixel wide representations (skeletons) of the original character image. The preliminary image contains individual character templates strung together to form a string. Since the textlines have to be close to human handwriting style, important aspects like character baseline alignment, scaling of character sizes and ligature joins have to be considered. We present original techniques for automatic baseline detection and ligature handling. We also present a character auto-scaling technique. Once we have the preliminary image, we apply a set of geometric, image level perturbations that distorts the preliminary image in a random fashion. The perturbations can be parameterized and this allows us to pick random values over a range of values. The technique follows the model presented in [15]. Finally, the distorted image is thickened. Thickening can be controlled so that different parts of the image are thickened by different amounts.

A dataset of over 20,000 character images[1] which contains multiple handwritten samples of each English character has been used. The characters have been segmented out manually from actual pieces of US mail. For a large fraction of the cases, the beginning and ending ligatures are also present with the character. Fig. 1 shows some examples of the character images.

We first construct a preliminary image, which is a concatenation of individual character templates. Character templates are one-pixel wide representations (skeletons) of the original character image. We use Blum's medial axis transform [20] to generate the character templates.

4.1. Character auto-scaling

To form the preliminary image, we need to concatenate individual characters to form the required textline. We perform auto-scaling of the characters so that all characters maintain their correct relative sizes with respect to each other. This means that, for instance, a 'p'

[1] Dataset collection and character segmentation done by CEDAR, University at Buffalo, NY, USA.

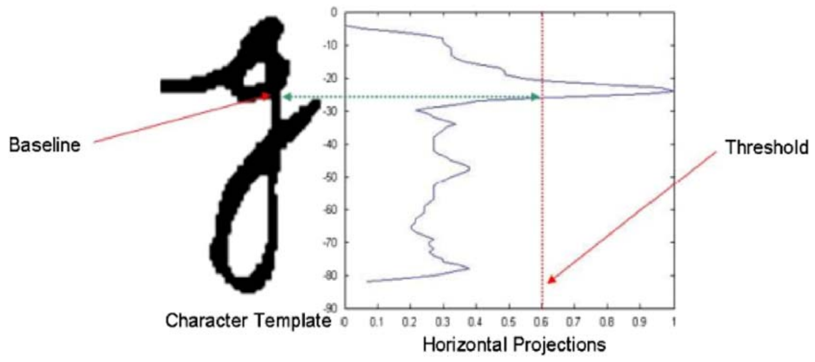

Fig. 2. Automatic baseline determination for character image 'g'.

has to be twice as long and once as wide as an 'a'. Auto-scaling is a required step since different samples of the same character could be of different sizes or the relative sizes of distinct characters could be incorrect. We base the auto-scaling on the absolute size of the first character in the string. We maintain a lookup table of character heights. This is known as the scaling factor for a character sf_c. For instance, the scaling factor for 'a' is 1, 'b' is 2, 'c' is 1 and so on. The scaling factor gives the number of segments of a three segment space occupied by a particular character. The height for character i is calculated as $(sf_{fc}/sf_i) * h_{fc}$ where sf_{fc} is the scaling character for the first character, sf_i is the scaling factor for character i and h_{fc} is the height in pixels of the first character. The character width is scaled relative to the character height as per the original aspect ratio.

4.2. Automatic baseline determination

To string together individual characters to form a textline, we need to make sure that the characters are aligned vertically at their true baselines. We have exploited the fact that, in our dataset, the ligatures give us clues regarding the location of the true baseline. The procedure to determine the true baseline is as follows. Sum up the horizontal projections of n consecutive rows and store this value sHP_R for row R. Thus, $sHP_R = \sum_{r=R-n}^{R} HP_r$ and HP_r is the horizontal projection value for row r. Normalize $sHP_{Rows \ldots n+1}$ so that $sHP_{Rows \ldots n+1} \in (0, 1]$ and declare the true baseline as row TB where $TB = first(sHP_{Rows \ldots n+1} > threshold_B)$ and $Rows$ is the total number of pixel rows in the character image, $threshold_B$ is a cut-off value that is determined empirically from the dataset. The function $first(\,)$ returns the first value, in some ordered set X, that matches some given criteria. $first(\,)$ looks from the bottom-most horizontal projections to the top-most horizontal projections. Fig. 2 shows a sample character image for 'g' and the corresponding horizontal projections; n was taken as 2 in this case and $threshold_B$ was determined to be 0.75 empirically. Once a character dataset is fixed, the values of n and $threshold_B$ need not be altered.

The reasoning behind using this approach was arrived at after noticing that the ligatures of the character join the main body of the character at the true baseline. For cases where this is not true, like 'o', using the $first(\,)$ function solves this problem, since we look for the first horizontal projection from the bottom upwards that crosses the threshold. We need to consider more than just a single horizontal row while calculating the sums of horizontal projections. This is because the ligature need not be perfectly horizontal, but it is also always is inclined at only a small angle around 0. Using $n > 0$ takes care of this issue as we are considering consecutive rows of horizontal runs of pixels.

4.3. Ligature endpoint detection

Ligature handling is an important part of the procedure since ligatures make a textline look like human handwriting. Several approaches have been proposed for handling ligatures in synthetic handwriting generation [11,12,14]. However, these approaches have been geared towards writer-specific handwriting generation. For our application, we decided to use the already existing ligature information and extend from that point on. We need a way to parameterize the ligature that connects to the body of the character. To achieve this, we need to extract the start and end ligatures from the image. Following a procedure similar to the automatic baseline detection method, we compute a statistic known as the pseudo-inking profile from the image. The pseudo-inking profile captures the pen activity as the writer generated the character image. We compute the sums of vertical projections of n consecutive columns and store this value sVP_C for row C where $sVP_C = \sum_{c=C-n}^{C} VP_c$ and VP_c is the vertical projection value for row c. We now consider the first derivative of the pseudo-inking profile. We define the first derivative $fdsVP_C$ as $fdsVP_C(i) = sVP_C(i+1) - sVP_C(i)$, where$i \in [1 \ldots C-1]$. To detect the ligature endpoints we consider the left and right half images of the character separately to perform normalization of the first derivative plot. Fig. 3 shows how the first derivative plot is divided and normalized separately. We can use the first derivative plots to detect the ligature endpoints by defining a threshold value $threshold_L$. We now look in the normalized half plots of the first derivative for the first peak above some threshold. We compute $ligature_B = first_{1 \ldots C}(fdsVP_{1 \ldots C/2} > threshold_L)$ and $ligature_E = first_{C \ldots 1}(fdsVP_{C/2 \ldots 1} > threshold_L)$. These give us the columns where the beginning and ending ligatures join the character body. We empirically decide on a single global $threshold_L$ based on the dataset.

The reason for not splitting the first derivative plot exactly down the center is because for characters like 't' and 'i' with a very thin character body, it is important that the character body information is included while normalizing the half plots. Fig. 4 shows the correctly detected ligature points for character images 'i', 'o', and 'd'. Our tests have proven that the ligature points have been detected correctly even for 'd' which has hardly any beginning ligature.

4.4. Ligature parameterization

We determine the points at which the ligatures join the main character body. For some characters, such as 'f' or 't', more than one ligature is possible. We use heuristic rules to decide on which ligature to use, either we can pick the largest/smallest ligature component or pick one component at random. We have used a random pick in our method.

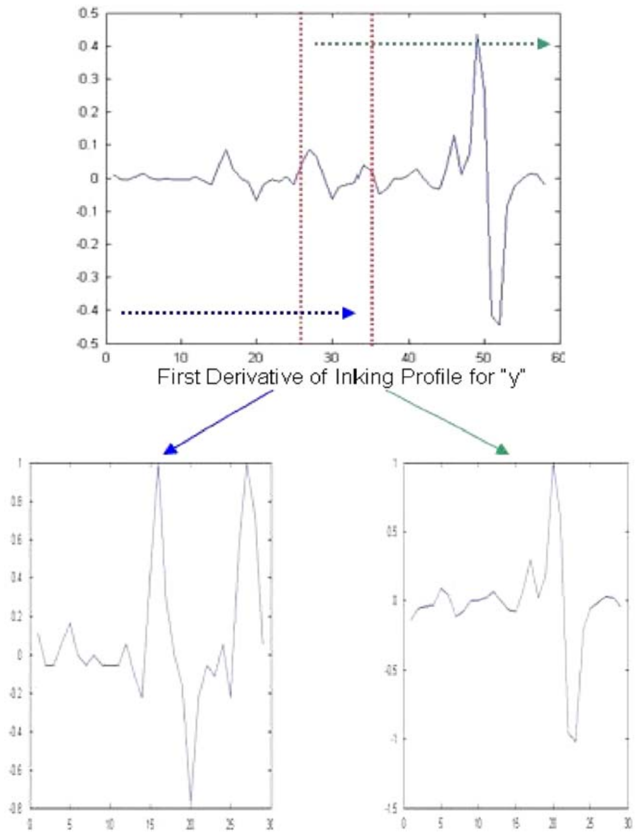

Fig. 3. First derivative plot is split after allowing overlap and then normalized.

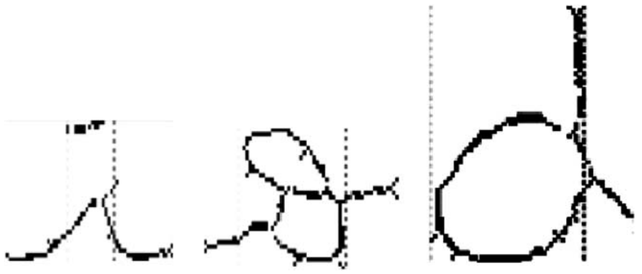

Fig. 4. Correctly detected ligature points for 'i', 'o' and 'd'.

To fit an nth order polynomial $p(x) = a_n x^n + a_{n-1} x^{n-1} \cdots a_1 x + a_0$ to the extracted ligature, we use regression to perform polynomial approximation. To determine the order of the polynomial to represent the ligature, we first start with a polynomial of order 1, determine the polynomial and then compute a reconstruction error. The reconstruction error is defined as the number of mismatched points between the original ligature and the reconstructed ligature. We move to progressively higher order polynomials until the reconstruction error stabilizes and then choose the polynomial with the lowest reconstruction error. Fig. 5 shows the character images for 'o', 't' and 'a' and the corresponding parameterized ending ligatures. The beginning ligatures can be parameterized in a similar manner.

4.5. Ligature joining

Given the parameterized ligatures, we can once again use a set of heuristic rules to join the ending ligature of character i to the starting ligature of character i+1. We can just extend the ending ligature of character i to just touch character i+1 or vice-versa with the starting ligature of character i+1. Another approach would be to assign strengths to the ligatures and decide on the join based on the dominant ligature. A simple measure of ligature strength could be the number of pixels in the ligature. A better approach would be to ensure a smooth join from the ending ligature of character i to the starting ligature of character i+1. We use the following approach for ligature joining. Consider only the ending ligature of character i. Discard the beginning ligature of character i+1. Align character i+1 at a distance of $d/2$ (where d is the width of character i+1) from character i. Now trace the ending ligature of character i to continue till it joins character i+1. Since character i+1 has no beginning ligature, the traced stroke from character i joins naturally to character i+1.

4.6. Skeleton perturbation

We use the perturbation model proposed in [15,16] for the distortion of cursive handwritten textlines. This is needed so that each time, the generated textline will appear different. The perturbations take the form of a set of geometrical transformations. Each geometrical transformation is controlled by a continuous nonlinear function, (underlying function), which determines the strength of the transformation at each horizontal or vertical coordinate position of the textline. The underlying function is typically composed of piecewise sinusoidal functions which are stitched together to form a continuous function and they control geometrical transformations that affect a whole line of text. The perturbation model incorporates a set of parameters over a range of possible values, from which a random value is picked before an existing textline is distorted. The range of values for these parameters is chosen such that the maximum perturbation just places the distorted textline at the edge of human readability. The transformations include shearing, horizontal and vertical scaling, and baseline bending. A detailed description of the perturbation model is available in [15,16].

4.7. Skeleton thickening

At this point, the textlines are still single pixel wide images. To thicken the textlines, we use the inverse medial axis transform. To add to the image complexity, we also apply an underlying function to vary the stroke width across the textline. Examples of final synthetic handwritten images are shown in Fig. 6.

5. CAPTCHA distortion

The primary application of our synthetic handwriting generator is automatic random generation of infinitely-many distinct handwritten (image) challenges for cyber security. Our ability to generate infinitely many handwritten word images implies that we are not limited by a finite size database of CAPTCHA challenges. This makes the method suitable for online applications. Once a handwritten word image or textline has been generated, we add further distortions to obfuscate the image to a greater extent. Fig. 7 shows some examples of the kind of distortions that we have used. Note that these are just an arbitrarily chosen set of distortions. It is certainly possible to think of other distortions. What follows is a brief description of the different distortions we have used.

Edge: The image edge contour is traced by applying the canny edge operator to the word image. An instance of this distortion can be seen in word "*aesthetic*" in Fig. 7.

Fragmentation and displacement: First a vertical row number is randomly chosen, subject to the condition that the row is located between the ascenders and descenders of all characters. For the fragmentation distortion, the top part of the image is shifted horizontally randomly to the right or the left. For the displacement

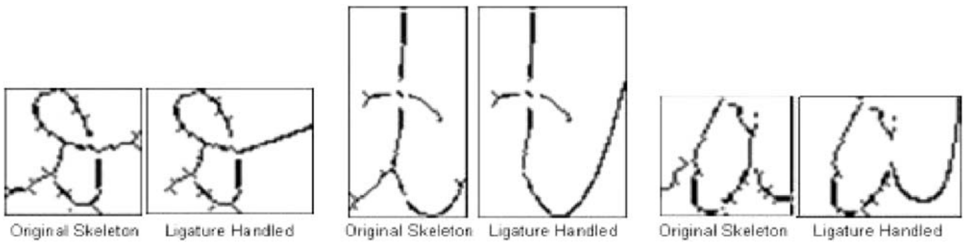

Fig. 5. Parameterized ending ligatures for 'o', 't', 'a'. Original images are on the left, processed images are on the right.

distortion, the top part of the image is shifted vertically by a random amount first, and then horizontally shifted. The horizontal shift value is constrained to lie between 0.25 and 0.75 times the average character width. An instance of this distortion can be seen in the word "*anatomy*" in Fig. 7.

Jaws/arcs and waves: A continuous nonlinear sinusoidal curve is generated using the perturbation model explained in Section 4.6. For the jaws/arcs distortion, this curve is overlaid on the word image using the foreground color. For the waves distortion, this curve is overlaid on the word image using the background color. An instance of this distortion can be seen in the word "*abort*" in Fig. 7.

Mosaic: A set of randomly oriented lines are overlaid on the word image at random locations, using the background color. The line width is constrained to match the stroke width of the characters.

Circles: A set of disks of random sizes are overlaid on the word image at random locations, using the background color. The circle diameter is constrained to lie between 0.25 and 0.75 times the average character width.

Exploded: First, the word contour is obtained as in the edge distortion. Next, each contour pixel is randomly offset. Larger the offset, larger the distortion effect produced. An instance of this distortion can be seen in the word "*abandon*" in Fig. 7.

Overlap: A copy of the word image is overlaid over the original image, shifted by a vertical, horizontal or combined offset. The shift value is constrained to lie between 0.25 and 0.75 times the average character width. An instance of this distortion can be seen in the word "*breeding*" in Fig. 7.

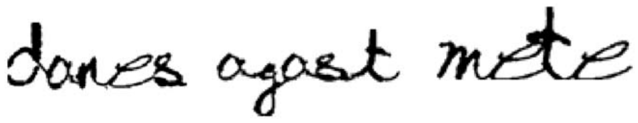

Fig. 6. Synthetic handwritten word images.

6. Performance evaluation

In this section, we look at the performance of the handwritten CAPTCHA as perceived by humans and handwriting recognizers (OCRs). Two recognizers were used, Word Model Recognizer (WMR) [21] and Accuscript [22]. WMR is a segmentation-based recognizer. It considers each word to be a model and finds the best match between an entry in a lexicon and the image. Accuscript is a grapheme-based recognizer. It extracts features from sub-characters (loops, turns, junctions, arcs, etc.) without explicit segmentation. Both recognizers take advantage of using static lexicons in the recognition process, as well as using pre-processing techniques to enhance image quality and remove noise, thus making the performance evaluation for machines a fair test. The recognizers were run on a set of 2800 generated images distributed evenly among the various types of distortions.

One could argue that using lexicon based recognizers is not representative of the real world scenario for present day CAPTCHA implementations, because we reduce the problem to include only a limited number of challenges. However, for the tests with human subjects, we chose to only generate words that occur in the English language. The reason for this can best be described as seen in Fig. 8. For handwritten word images, character formation ambiguity arising from different writing styles can lead to multiple segmentation hypotheses for a given word image. As seen in Fig. 8, three segmentation hypotheses are possible for the handwritten word. Humans (and recognizers relying on lexicons), decide on the correct segmentation hypothesis based on either context information, or knowledge of the lexicon. Human recognition accuracy would decrease if the handwritten CAPTCHA challenges were allowed to be lexicon independent. This defeats the purpose of using handwriting for CAPTCHAs. So, for the tests, we limit ourselves to a lexicon that comprises all the words in the English language. This makes it feasible for humans to use context information (that the word is a legitimate word in the English language), to guess the handwritten word.

Table 1 presents the recognition accuracy of two types of OCRs on CAPTCHA challenges with different types of distortions. Even with no distortions applied on the generated handwritten word images, the recognizers are not able to cross the 40% mark. When distortions are applied, the recognition rate drops much below 10%. The average recognition rate is comparable for the two recognizers with Accuscript performing marginally better.

Our proposed technique improves on prior work published in 2006 [19]. As reported in Table 2, for all distortions, the highest machine accuracy was about 13% as opposed to the current, much lower, 2.6%. Note that the lower the score, the better, when considering machine accuracy.

Table 3 presents the corresponding human performance on the test set. For human testing, random CAPTCHA samples were presented to users through a website. 2800 responses were collected from around 100 users. With no distortions, the average recognition rate for humans is about 84%. The recognition rate drops for other distortions. These results along with results in Table 1 are helpful in determining which types of distortions would be more suitable for use in CAPTCHAs. For all distortions, the average human recognition rate is about 76%, which is the same as in the prior work of Rusu and Govindaraju [19]. Thus, human recognition rate has not decreases even though the machine recognition rate has been considerably lowered by our proposed method.

An interesting test to perform is to compute machine recognition rates on pre-processed versions of the distorted CAPTCHA images. The pre-processing would try to undo the distortions applied to the generated CAPCTHA images. This is a reasonable approach as we must assume that hackers will be sufficiently motivated to develop algorithms to undo image distortions, before using a recognizer on the textlines. Intuitively, it would seem that such pre-processed images would be easier for machines to recognize. To this end, we invited a programmer to develop algorithms to try and undo the

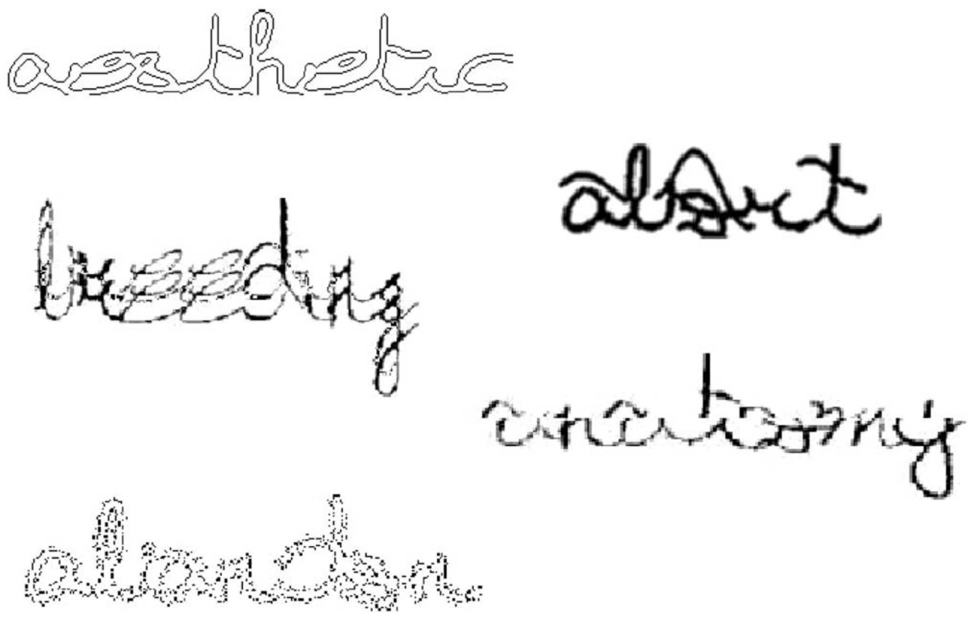

Fig. 7. Different types of distortions applied to generated handwritten word images.

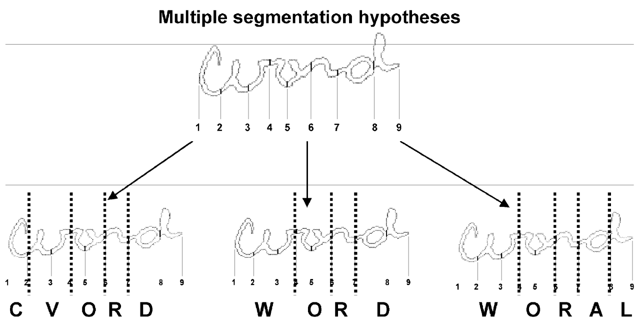

Fig. 8. Multiple segmentation hypotheses for a handwritten word.

Table 1
Recognition accuracy of OCRs for different CAPTCHA distortions.

HW recognizer (OCR)	WMR (%)	Accuscript (%)
No distortion	23.69	37.78
Perturbed image	0.18	2.63
Edges	0.18	0
Fragmentation	1.43	3.45
Displacement	0	3.61
Mosaic	0	3.78
Jaws/arcs	5.91	5.83
Occlusion by circles	0.36	5.75
Occlusion by waves	0	2.30
Exploded image	0	0
Vertical overlap	1.35	1.32
Horizontal overlap	4.91	1.16
Sideways overlap	2.69	1.16
All distortions	1.42	2.58

Table 2
Comparison with prior work-Machine accuracy.

HW recognizer	WMR (%)	Accuscript (%)
All distortions (IWFHR 2006)	12.7	6.4
All distortions (current)	1.42	2.58

Table 3
Recognition accuracy of humans for different CAPTCHA distortions.

Human performance	Accuracy (%)
No distortions	83.97
Perturbed image	77.24
Edged image	78.08
Fragmentation	84.17
Displacement	77.55
Mosaic	69.53
Jaws/arcs	69.96
Occlusion by circles	71.71
Occlusion by waves	84.25
Exploded image	76.62
Vertical overlap	74.45
Horizontal overlap	80.77
Sideways overlap	66.67
All distortions	76.29

distortions applied to the generated CAPTCHA images. The programmer was given the generation method for the CAPTCHA images and the subsequent distortions applied to the generated textlines and allowed access to a number of sample images, from all classes of distortions. This allows us to test what the machine recognition rates would be if the adversary had knowledge of the CAPTCHA generation procedure.

Table 4 presents the machine performance on these pre-processed CAPTCHA images for a subset of the distortions. The recognition rates, for all but one distortion type, are comparable to the original distorted images. This shows that the distortions obfuscate the images by a considerable amount while still maintaining readability for humans. This test allows determining what kinds of distortions are better for use in CAPTCHAs.

Fig. 9 shows the clear gap between human and machine recognition abilities for handwritten CAPTCHAs. This shows that handwriting has potential for use in CAPTCHA applications. We also see that humans perform better at recognizing images with certain type of distortions (waves, fragmentation, horizontal displacement) over others. For these distortions, machine performance is low for both the original distorted image case as well as the pre-processed image case. Hence, these types of distortions would be more successful for use in CAPTCHAs.

Table 5 gives the success rates at breaking a number of popular printed text CAPTCHAs as mentioned in [3]. For the handwritten CAPTCHAs presented in this work, the success rates for machines are much lower.

7. Discussion

For suitable use in a cyber security scenario, we must ensure that the human recognition rate stays high when compared to machine recognition rates (which should ideally be 0). Even a seemingly low recognition rate for machines (say, less than 0.001%), would not necessarily mean that a given cyber security application can be considered bot-proof. We must bear in mind that a recognition rate of x% means that, statistically, x out of every 100 attempts will be successful. Since it is possible to have distributed attack networks, repeatedly trying to gain access to a secured application, the shear volume of attacks would render the low recognition rate itself, irrelevant. This means that a human verification system using CAPTCHAs, (handwritten or otherwise), needs to ensure that additional checks and measures are in place to handle the rapid repeated attacks on the verification service. A simple test would be to check from where the verification requests are originating. Requests arriving within t seconds of each other, from the same IP address can be denied. In practice, t would be set to a small value to discourage web-bots that try to repeatedly access a CAPTCHA protected site. Each IP can only be allowed n number of authentication requests in some pre-determined time period. These and other methods (possibly inspired from the computer security field) need to be used in conjunction with CAPTCHAs, while designing effective automated human verification systems. Relying on CAPTCHAs alone to solve the human verification problem is unrealistic. As artificial intelligence algorithms get better every day, and given the superior processing capabilities of machines, a 0% recognition rate cannot be guaranteed.

On the other hand, while it would be ideal to have 100% recognition accuracy for human users, it would be safe to assume that human users would tolerate some lack of CAPTCHA ease. For instance, a human user might not be overly concerned with having to re-try a CAPTCHA once in every 8–10 attempts. This fact can be exploited while designing CAPTCHAs. Some human recognition performance can be deliberately sacrificed, if it degrades machine performance by a considerable amount.

CAPTCHAs have been developed to deal with the specific case of machines trying to masquerade as humans. Web services that require human verification only focus on differentiating between machines and humans. Such types of web services are very vulnerable to attacks from humans itself. This fact has led to some concern of having CAPTCHAs broken by humans in what is known as the *Pornography Attack*. The CAPTCHA challenge encountered by a web-bot is forwarded to human users on some high traffic website (e.g. a pornographic website). The human users are required to decode the CAPTCHA to get access to pornographic images. The decoded responses are relayed back to the web-bots who supply the response to the web service and gains access to the service. The official CAPTCHA website at [2] describes how the above mentioned scheme will not make economic sense under the paragraph *The "Pornography Attack" is not a concern.*

Table 4
Machine performance on pre-processed handwritten CAPTCHA images.

HW recognizer	WMR	Accuscript
Lexicon size	4000	4000
Edges	13.14%	23.97%
Fragmentation	0.99%	0.0%
Exploded image	4.11%	0.49%
Vertical overlap	0.0%	2.96%
Horizontal overlap	0.82%	0.33%
Sideways overlap	0.0%	0.33%

Table 5
Success rate in breaking various popular printed text CAPTCHAs.

CAPTCHA	Mailblocks	Register	Yahoo! v2	TicketMaster	Google
Success rate (%)	66.2	47.8	45.7	4.9	4.89

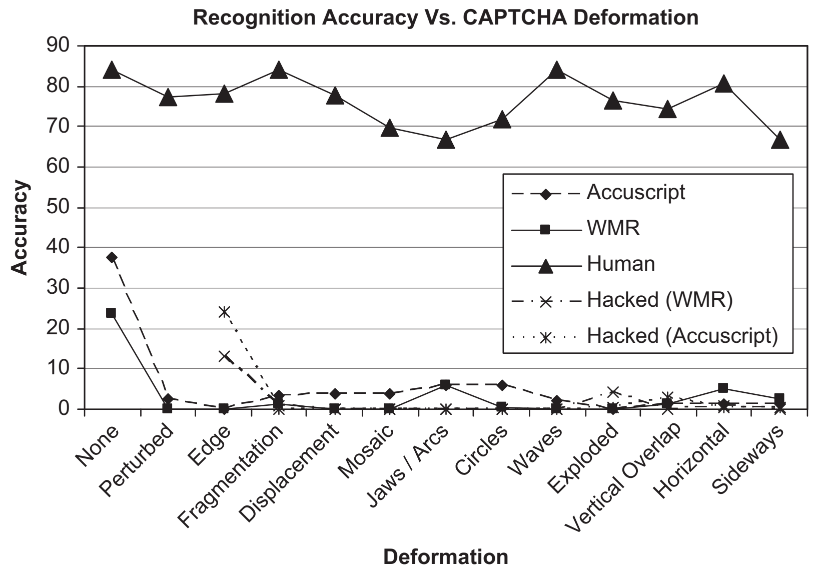

Fig. 9. Gap in recognition abilities between humans and machines for the handwritten CAPTCHA. The hacked WMR and Accuscript plots are for the pre-processed images.

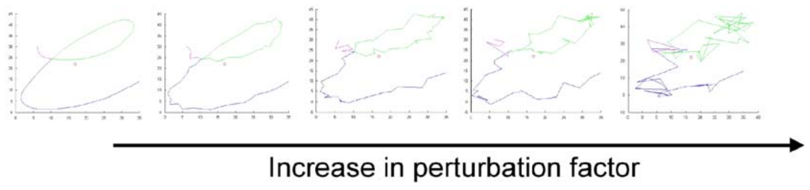

Fig. 10. Character template for the 'e' at varying levels of perturbation.

8. Conclusion and future work

We have explored the recognition performance of humans as compared to machines, on handwritten CAPTCHAs. We have shown how to generate synthetic handwriting samples and then apply various distortions to make them near-unreadable by automatic computer programs, so that they can be used as CAPTCHA challenges over the Internet. Test results show a large gap between human and machine recognition abilities. There is also a significant decrease in machine recognition rates for our synthetic samples as compared to prior work. However, human recognition rates remain the same. This directly translates as the method being a better CAPTCHA generation technique.

We are currently in the process of performing human perception tests on printed character CAPTCHAs and comparing them against machine recognition rates. We plan to improve the proposed method by automatically learning the threshold values $threshold_B$ and $threshold_L$ from a given dataset of character images. We plan to research on deformation techniques that exploit the knowledge of the common source of errors in automated handwriting recognition systems and also take advantage of the cognitive aspects of human reading. We plan to conduct more detailed tests that will involve using textlines that are generated by stringing together random characters and also words formed by concatenating two or more phonemes using a phonetic generator [9]. If the human recognition performance is comparable to the current results, it would mean that we can further reduce the machine recognition performance since the lexicon size will increase considerably.

As an extension to using pre-handwritten character samples, we plan to use character templates in the generation of the texlines. Kegl and Krzyzak [23] describes a method to construct piecewise linear skeletons of handwritten characters using principle curves. The character templates are represented as a set of control points. This approach has the advantage of allowing distortions at the character level itself by perturbing the control points. The perturbations can be parameterized. Fig. 10 shows the character template for the handwritten character 'e' and varying levels of perturbation applied to the template. As the perturbation factor increases, the character becomes more illegible. An important aspect will be controlling the perturbations. The maximum limit of the perturbation parameter can be determined empirically by performing human perception tests. Once the limit is known, the distortion can be varied within that limit and human and machine accuracies can be determined for varying perturbation levels. This would make it possible to generate operating curves. Picking a perturbation value p will give us the accuracies that humans and machines will have at perturbation level. CAPTCHAs can be generated for various security levels using this approach.

Another application of the handwriting generator is to improve the accuracy of handwriting recognizers by generating large synthetic training data sets. Since our technique does not generate writer-specific handwritten textline samples, we could use it for training generic handwriting recognizers.

Acknowledgments

We would like to thank Philip Kilinskas, Research Scientist at CEDAR, for developing the algorithms to reverse the CAPTCHA distortions.

References

[1] A.M. Turing, Computing machinery and intelligence, Mind 59 (236) (1950) 433–460.
[2] The official CAPTCHA website ⟨http://www.captcha.net/⟩.
[3] K. Chellapilla, K. Larson, P.Y. Simard, M. Czerwinski, Building segmentation based human-friendly HIPs, in: Proceedings of Human Interactive Proofs, Second International Workshop (HIP 2005), Springer, Bethlehem, PA, USA, 2005.
[4] ⟨http://www.captcha.net/cgi-bin/esp-pix⟩.
[5] ⟨http://www.toallwhoseekit.net/cgi-bin/sq-pix⟩.
[6] A.L. Coates, H.S. Baird, R.J. Fateman, Pessimal print: a reverse Turing test, in: Proceedings of Sixth International Conference on Document Analysis and Recognition (ICDAR 2001), IEEE Computer Society, Seattle, WA, USA, September 10–13, 2001, pp. 1154–1158.
[7] H.S. Baird, M.A. Moll, S.Y. Wang, A highly legible CAPTCHA that resists segmentation attacks, in: Proceedings of Human Interactive Proofs, Second International Workshop (HIP 2005), Springer, Bethlehem, PA, USA, 2005, pp. 27–41.
[8] A. Rusu, V. Govindaraju, A human interactive proof algorithm using handwriting recognition, in: Proceedings of Eight International Conference on Document Analysis and Recognition (ICDAR'05), IEEE Computer Society, 2005, pp. 967–971.
[9] M. Chew, H.S. Baird, Baffletext: a human interactive proof, in: Proceedings of SPIE/IS&T Document Recognition and Retrieval X Conference, San Jose, CA, USA, January 2003.
[10] H.S. Baird, K. Popat, Human interactive proofs and document image analysis, in: Proceedings IAPR 2002 Workshop on Document Analysis Systems, 2002.
[11] Z. Lin, L. Wan, Style preserving English handwriting synthesis, Microsoft Research Asia, 2005.
[12] J. Wang, C. Wu, Q.-Y. Xu, H.-Y. Shum, Combining shape and physical models for online cursive handwriting synthesis, in: International Journal on Document Analysis and Recognition, 2004, pp. 2097–2109.
[13] W. Guerfali, R. Plamondon, The delta log normal theory for the generation and modeling of cursive characters, IEEE, September 1995
[14] M. Kokula, Automatic generation of script font ligatures based on curve smoothness optimization, Electronic Publishing 7 (4) (1994) 217–229.
[15] T. Varga, H. Bunke, Generation of synthetic training data for an HMM-based handwriting recognition system, Proceedings of the Seventh ICDAR, Edinburgh, Scotland, 2003, pp. 618–622.
[16] T. Varga, H. Bunke, Effects of training set expansion in handwriting recognition using synthetic data, in: Proceedings of the 11th Conference of the International Graphonomics Society (IGS) 2003, Scottsdale, AZ, USA, November 2003, pp. 200–203.
[17] M. Mori, A. Suzuki, A. Shio, S. Ohtsuka, L.R.B. Schomaker, L.G. Vuurpijl (Eds.), Generating new samples of handwritten numerals based on point correspondence, in: Proceedings of the Seventh International Workshop on Frontiers in Handwriting Recognition, Amsterdam, ISBN 90-76942-01-3, Nijmegen: International Unipen Foundation, 11–13, September 2000, pp 281–290.
[18] H.S. Baird, Anatomy of a versatile page reader, Proceedings of the IEEE 80 (1992) 1059–1065.
[19] A. Rusu, V. Govindaraju, The influence of image complexity on handwriting recognition, in: Proceedings of International Workshop on Frontiers in Handwriting Recognition, 2006.
[20] H. Blum, A transformation for extracting new descriptors of shape, in: W. Wathen-Dunn (Ed.), Models for the Perception of Speech and Visual Form, MIT Press, Cambridge, MA, 1967, pp. 362–380.
[21] J.T. Favata, Character model word recognition, in: Proceedings of the Fifth International Workshop on Frontiers in Handwriting Recognition, 1996, pp. 437–440.

[22] H. Xue, V. Govindaraju, A stochastic model combining discrete symbols and continuous attributes and its applications to handwriting recognition, in: Proceeding of the International Workshop on Document Analysis and Systems, 2002, pp. 70–81.

[23] B. Kegl, A. Krzyzak, Piecewise linear skeletonization using principal curves, IEEE Transactions on Pattern Analysis and Machine Intelligence 24 (1) (2002) 59–74.

About the Author—ACHINT OOMMEN THOMAS completed his MS in Computer Science and Engineering from the University at Buffalo (SUNY) in 2007 and is currently a PhD candidate there. His areas of interest include HIPs, CAPTCHAs, pattern recognition and biometrics.

About the Author—AMALIA RUSU is an Assistant Professor in the Department of Software Engineering, School of Engineering at Fairfield University. She received her PhD in Computer Science and Engineering from University at Buffalo (SUNY) in 2007. Her main research focus is on handwriting recognition and CAPTCHAs.

About the Author—VENU GOVINDARAJU is a Professor of Computer Science and Engineering at the University at Buffalo (SUNY). He received his B-Tech (Honors) from the Indian Institute of Technology (IIT), Kharagpur, India, in 1986, and his PhD from SUNY Buffalo in 1992. His is also the Founding Director of the Center for Unified Biometrics and Sensors (CUBS) and the Associate Director of the Center of Excellence for Document Analysis and Research (CEDAR), both at SUNY Buffalo.