

Decoding the animated text-based captchas to verify their robustness against automated attacks

Rafaqat Hussain Arain ^{a, *}, Riaz Ahmed Shaikh ^a, Safdar Ali Shah ^a, Sajjad Ali Shah ^a, Saima Rafique ^b, Ahmed Masood Ansari ^a

^a *Institute of Computer Science, Shah Abdul Latif University, Khairpur Sindh Pakistan*

^b *Government Girls Degree College, Pir Jo Goth, Sindh Pakistan*

* Corresponding author: Rafaqat Hussain Arain, Email: rafaqat.arain@salu.edu.pk

Received: 01 August 2023, Accepted: 25 September 2023, Published: 01 October 2023

KEYWORDS

Image Processing
Machine Learning
Artificial Intelligence
CAPTCHAs
Web Security
Pre-trained CNN

ABSTRACT

In order to protect the web against automated attacks, CAPTCHAs are most widely used mechanism on the internet. Numerous types of CAPTCHAs are introduced due to weaknesses in the earlier designs. Animated CAPTCHAs are one of the design alternatives. Instead of presenting the whole information at once, animated CAPTCHAs present information in various frames over the specific interval of time. As CAPTCHAs are ubiquitously used to avoid the serious threats from bots therefore it is important to verify their effectiveness. In this research we have verified their robustness against machine learning attacks. It has been proved that adding the extra time dimension does not necessarily ensure protection against automated attacks. We have attacked the Hello CAPTCHA scheme, which is the most popular animated CAPTCHA scheme available on the internet. By applying novel image processing and machine learning techniques, these CAPTCHAs are decoded with high precision. A pre-trained CNN is used to recognize the extracted characters. In this research, 6 popular types of animated CAPTCHAs along with 41 sub types were successfully deciphered with an overall precision of up to 99.5 %.

1. Introduction

CAPTCHA (Completely Automated Public Turing Test to tell Computers and Human Apart) is a method to safeguard the web against bots. This is most accepted and extensively used mechanism over the internet [1]. Although, various design variants including image, audio, video and other various types of CAPTCHAs were introduced over the years, but still text-based CAPTCHAs are most commonly used owing to their simplicity and ease of use [17, 19]. Since its introduction, various researchers attempted to verify its robustness. Initial CAPTCHAs were found vulnerable against automated attacks and decoded by many researchers with high precision [2, 3, 7]. As the CAPTCHAs were attacked and

successfully decoded, the designers continued their effort to make them more robust. In this way, more and more new design variants were introduced in last couple of decades. Initial text-based CAPTCHAs were simple, but their design variants included more noise, cluttering and overlapping mechanisms [20, 21]. However, as the new designs were introduced, new methods were designed to break them. Therefore, it became an ongoing war between designers and attackers. It is a healthy competition between both, because in either case it is useful. In case, if a CAPTCHA is designed and not yet broken then it can be implemented as a security mechanism. Otherwise, if it is solved then it may lead to one step forward in the field of AI (Artificial Intelligence). As CAPTCHAs are based on difficult problems in the

field of AI therefore if these CAPTCHAs are broken then the breaking algorithm can be used to move forward in AI.

Animated CAPTCHAs are also introduced in an attempt to make more robust CAPTCHAs. It is assumed that these CAPTCHAs may be more robust than the static CAPTCHAs. In animated CAPTCHAs the information is distributed in several frames [4-6]. Therefore, it is assumed that it is hard to be extracted by decoding algorithms. However, we attempted to break these CAPTCHAs and found serious vulnerabilities in them. The use of fixed time intervals, fixed locations for specific characters and simple font types are few design weaknesses which were exploited. On the basis of such weaknesses, we decoded these CAPTCHAs with high precision. It is concluded in the end that adding extra time dimension does not provide more security against programmed attacks.

In our research, it is found that very few animated CAPTCHA schemes are available on the internet. Among the found CAPTCHAs, Hello CAPTCHA is the most popular and widely available on the internet [4]. In this work we have selected 6 types along with 41 sub types of Hello CAPTCHAs. All types of Hello CAPTCHAs are successfully decoded by our proposed algorithms. Solving the CAPTCHAs begin with obtaining the animated image and converting it to static image using specific algorithm. The animated image consists of multiple frames containing partial information in each frame. All these frames are extracted and the information in terms of foreground text is obtained using our proposed methods. Preprocessing operations are performed on the obtained static image. Furthermore, the image enhancement, noise removal, and certain image processing methods are applied. The obtained segments of characters may contain connected or non-connected characters which are further examined; segmented and machine learning techniques are applied to recognize them.

Section 2 presents literature review, section 3 consists of proposed methods, section 4 provides details of breaking all types of Hello CAPTCHAs and section 5 provides results and discussions while section 6 includes the conclusion and future work.

2. Literature Review

Athanasopoulos and Antonatos were the first to suggest the incorporation of animation as a method to enhance both the strength and usability of CAPTCHAs [9, 21]. This resulted in the design of animated text-based CAPTCHAs, in which

characters are moved independently in various vertical columns within the animated image [8]. It was thought that the addition of time would deliver greater security than usual static CAPTCHAs. It was referred as "zero knowledge per frame" by Cui et al. as the information needed to solve a CAPTCHA isn't solely present in a single image [10, 11].

Hello CAPTCHA is widely used real-world CAPTCHA that employs the security approach of dispersing information across several animation frames [4]. NuCAPTCHA is a text-based CAPTCHA with animation, designed to counter segmentation [13]. It features characters that are merged together, forming a moving text that a user can view in the space. Bots, however, don't have the same advantage and instead see a blurry mix of pixels [12]. The KillBot Professional CAPTCHA is the most complex of the initial CAPTCHAs which were based on motion. It uses a sophisticated combination of blurring and fading effects, as well as scaling, rotating, and moving characters up or down. It also integrates various animated noises such as raindrops and random characters which move horizontally or vertically, in addition to falling bars and moving lines [8]. Various types of animated text-based CAPTCHAs have been created by applying segmentation resistance principles, which usually overlap several animated characters within the frames in CAPTCHAs. The designers of NuCAPTCHA have developed a CAPTCHA based on animated text to counter segmentation attacks. While characters move left to right, each one is rotated separately. In addition, extra characters which do not rotate, along with an animated background, are included to make it hard for automated programs to decipher and capture the CAPTCHA. This dual complexity contributes greatly to the security of the CAPTCHA. At the University of Manchester, Kund created an animated CAPTCHA that involves small lines being used for two distinct types of moving elements. First type of CAPTCHA characters is composed of groups of particles, while the second type consists of single particles used as a background noise. These particles move in various directions to prevent bots from utilizing the PDM tracking method to detect the characters [14]. Xu et al proposed the first usage of the "emerging image" method within a text-based CAPTCHA for their new edition of NuCAPTCHA. This technique merges two sets of gray scale frames by employing an equation to compute the gray scale value for each pixel in each frame [15].

$$P(x, y) = S(x, y)e^{A(x,y)/C}$$

For all pixel points (x, y) with a value greater than a given number T , the method creates an animated final set of frames by making them white in Frame F from the first set and Frame S from the second set. The animated characters' edges will be shown in blurred images with a noisy background [15]. Nguyen et al. demonstrated a way of attacking animated text-based CAPTCHAs by applying technology to capture their animated figures in high-quality images. The attack was successful in breaking numerous early animated schemes. The approach used was a two-step process consisting of the CL (Catching Lines Method) and the PDM (Pixel Delay Map) [6], which allows for the extraction of characters that can be paused, moved, and scaled in individual columns. Research demonstrated that alternative approaches were effective exclusively when tackling certain animated CAPTCHAs during the character extraction stage. Frame Selection is specifically used on the animated CAPTCHA type with characters of the same colour that move in various directions and speeds. This technique involves selecting frames that have the greatest amount of pixels not originating from the background. Colour Selection is intended to be used on animated CAPTCHAs with characters of different colours that move in random directions and speeds. Roller Selection focuses on the animated CAPTCHA which contains rotated characters. Utilizing a flood fill algorithm, it segments these characters before extracting them by means of the CL method, which captures each single character when the highest point of its pixel reaches the highest possible height [16].

3. Proposed System

Our proposed system first identifies the type of animated CAPTCHAs and then applies specific methods according to the design and type of identified CAPTCHA. There are 6 types of Hello CAPTCHAs along with its 41 subtypes selected in this work to verify their robustness. Various design weaknesses were explored in this research, which includes time consistency, similar background in all frames, characters with no noise, and characters in all frames positioned at identical locations. Our proposed methods are applied to extract information from various frames. At first, a CAPTCHA challenge that is animated is converted into a single image, and then a specific method is used to break it. In this work, the key methods which are employed to break

these CAPTCHAs are: PCM (Pixel Changes Map), AF (Anti Fade), RCP(Remove Changing Pixels), CM(Catching Movers), GBI(Get Background Image), RF(Remove Footer), GP(Grace Pixel), RC(Replace Colours) and BI(Binary Image) method. By applying our proposed methods, all 6 types of animated Hello CAPTCHA schemes along with 41 subtypes are decoded with high precision. The proposed methods are shown in the system diagram in Fig. 1.

Once the image is loaded, which is typically an animated image (in GIF format); the system identifies its type. Generally, there are two categories of CAPTCHAs:

- (a) In type 1, individual characters stay at certain locations for 10 to 30 frames and then disappear. Flitters, H-Mover, Popup and Smarties are the type of CAPTCHAs which fall in this category.
- (b) In type 2, the CAPTCHAs are based on noisy backgrounds. Spread Fade and Searchlight are the type of CAPTCHAs, which fall in this category.

A pre-trained CNN (Convolutional Neural Network) is applied to initially identify the type of CAPTCHA provided. The CNN is trained by determining the mean sum of colours, and a score of 0 to 40 is assigned after the CAPTCHA is identified. Features such as the amount of frames and the background colours are used to identify the CAPTCHA. CNN is trained using 100 samples of each type of CAPTCHA and overall, 4100 images of all types of Hello CAPTCHAs are used to train CNN.

3.1 Extraction Techniques

Static images can be obtained from multiple frames of animated text by using extraction techniques. It is important to ensure that the characters in the resulting image are clearly visible and do not overlap, even if the image is very noisy. The techniques used to obtain a single image (in static form) from animated image are listed below:

PCM (Pixels Changes Map): Initially, the background of the first frame is selected (using our proposed GBI method). Once the background is obtained, it is compared to the other frames of the same challenge, and any changes in those frames are incorporated in the background image. This process is repeated for each subsequent frame, leading to a static image as illustrated in Algorithm No. 1.

Algorithm No. 1 (The PCM Method)

1. Set *ratio* to 255 / *frames*
2. Iterate through each *frame* in *frames*
 - Iterate through each pixel in *frame*
 - If pixel is not within background pixel *threshold*

$$Img \text{ pixel} += \text{ratio} * \text{abs}(\text{background pixel} - \text{frame pixel})/255$$
 - End inner loop
- End outer loop

The *img* argument can be used to salvage a still image from the animated image, while the array contained within the 'frame' argument holds all the frames of the CAPTCHA. The *threshold* is employed for altering the colors (background) of the image and for determining the values of it to eliminate the colors. The *background* argument is acquired using the GBI method.

AF (Anti Fade): This method is effective for breaking Spread Fade CAPTCHAs and their five

subtypes. These CAPTCHAs contain numerous small boxes of varying colors, with either white or black borders. This method looks for a part of the CAPTCHA that is not faded and then adds other such sections from consecutive frames, allowing for a static image to be re-created. The results of extracting the static image for Spread Fade/ immature dream CAPTCHAs are displayed in Fig. 2.

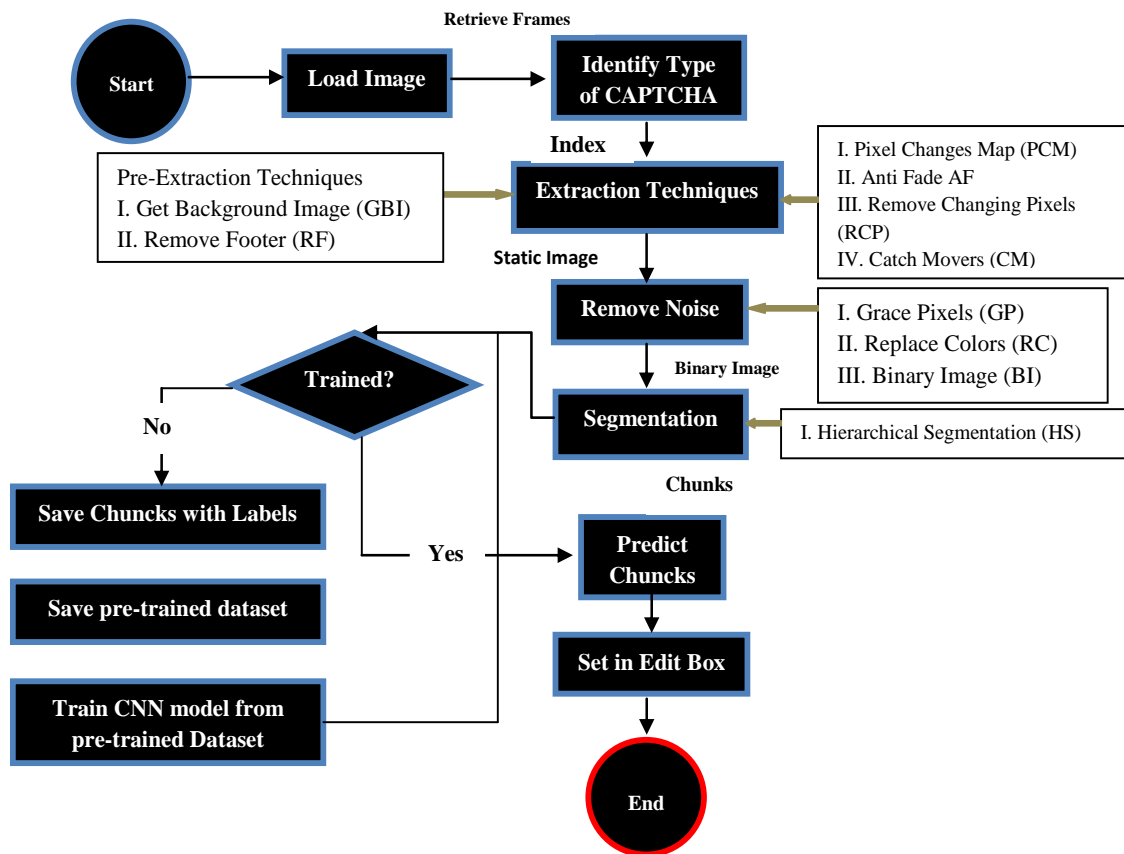


Fig. 1. The System diagram showing the proposed methods at different steps of decoding process

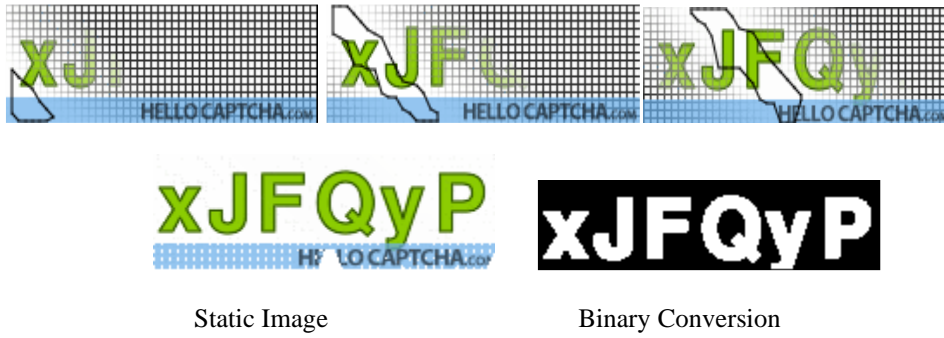


Fig. 2. Decoding SpreadFade/ Immature dream CAPTCHA by applying AF method

The steps required to get the static image are described in Algorithm No. 2. We can determine the width and height of each cell of the applied grid by identifying the area where the grid lines have been removed and storing the pixels in an array. A polygon of the stored pixels is then created, and the pixels are drawn in a static image within the polygon. The *out_img* argument stores the output of CAPTCHA characters without any fading, and the

un-faded area of the characters is established throughout all the frames. Additionally, the width of the boxes in the CAPTCHA is determined by the grid cell, and the colour of the boxes is stored in the grid colour. Inside this algorithm two functions are used: i.e., 'points2poly' and 'isPointInPoly'. The points in the array are used to create a polygon using the first function, i.e., points2poly while the unnecessary points are discarded.

Algorithm No. 2 (The AF Method)

1. Initialize *cwidth* to grid's cell width and *Gridcolor* to grid's line color
2. Iterate through frames
 - Iterate over *x* from *cwidth* to frame's width incrementing by *cwidth*
 - Iterate over *y* from 1 to frame's height incrementing by 2
 - Compare *Gridcolour* to *previousColour* and *nextColour*
 - If *Gridcolour* is equal to both *previousColour* and *nextColour* then push (*x*, *y*) to array of points
3. If length of points' array ≥ 3 then
 - call Points2Poly (points array) and assign the result to polygon variable
 - Iterate over *x* from 0 to frame's width
 - Iterate over *y* from 0 to frame's height
 - If isPointInPoly (*x*, *y*, polygon) returns true then
 - Set *out_img* color at (*x*, *y*) to frame color at (*x*, *y*)
4. Return *out_img*

After getting polygon the other function, i.e., 'isPointsinPoly' is used which takes three arguments; *x*, *y* and polygon to determine whether the pixels with coordinates *x* and *y* are within the polygon or not. In case of 'true' replace those pixels in static image. Using this method, the image without fading is retrieved.

colour is also black, the background *replacer* variable should be changed to white. It then compares each frame by examining the changes in the colour of individual pixels as given in algorithm No. 3.

RCP (Remove Changing Pixels): The RCP method is employed to decode search light CAPTCHA. The GBI process is used to start by finding out the background of the given animated picture. As a result, a clear image without text or noise is stored as a static image. Figs 3 (b, c, d) demonstrate the results of comparison while Fig. 3 (e) displays the final picture. If the value of background *replacer* is black and the background






Searchlight / Aunt blood CAPTCHA				
				
(a) Background	(b)	(c)	(d)	(e) Background with Text

Fig. 3. Decoding the Searchlight CAPTCHA by applying RCP method

It is hard to view the text in this type of CAPTCHA because the background and foreground colours are quite alike. To make the text visible, a

"Search Light" of a different colour is shaded over it, and the user can only see the text when the light is pointed directly at the foreground text.

Algorithm No. 3 (The RCP Method)

1. Set *out_img*' equal to *gbi*
2. Set *replacer* variable equal to black
3. If background colour from *gbi* argument equals black, then set *replacer* to white
4. Iterate through each frame in frames
 - a. Iterate through $x = 0$ to width of frame
 - b. Iterate through $y = 0$ to height of frame
 - If the colour of *out_img* at (x, y) is not equal to the frame colour then set *out_img* color at (x, y) equal to *replacer*.

CM (Catch Movers): CM method is employed to decode all sub-types of H-Mover CAPTCHAs. Experimental results suggest that the letters of the CAPTCHA can be found in the 140th to 179th columns, with the rightmost one being a new character. This technique will continuously scan the designated columns to detect a new character as soon as it appears. This process will repeat itself until all six characters have been obtained. The HS method (presented in this section) is applied to extort characters from animated frames, as displayed in Fig. 4. To calculate the number of foreground pixels per column in the image, the text or foreground is counted from the 10th position onwards. It is identified in three states: *come* (state = 0), *stay* (state = 1) and *leave* (state = 2). In order to get the animated characters by using the HS method from left to right, the state must be equal to *come*, and the frame is regarded as the actual frame. The state could *stay* in place until the character gets to the 140th column. The process of retrieving all six characters described in Algorithm No. 4 will begin.

after the 140th column. The state will be set to *leave* and the character will be treated as a new character. Afterwards, the state will be reset to *come* (0). This process will continue until all six characters have been retrieved.

GBI (Get Background Image): In Algorithm No. 5, the following steps are used to determine the background and footer of given image: first, the colour of all pixels in the first frame from the 15th row is checked. Then, the prevalent colour is considered as the background colour, and all other pixels from the 1st row to the 47th row (based on experiments) are altered to this colour. Next, at the 48th row, colour of all pixels is checked, and the prevalent colour is considered as the footer colour. Finally, all remaining pixels from the 48th row to the 60th row and 75th column are changed to the footer colour. The only argument this method takes is *frames*, which holds all the frames of CAPTCHAs. It returns the background image, free of noise and foreground text as shown in Fig. 5.

Frame 3 Frame 18 Frame 34 Frame 50 Frame 69 Frame 121



H-Mover/VioletSlider CAPTCHA



Replacing foreground and background



Segmentation using HS Method



Chunks obtained using CM Method

Fig. 4. H-Mover/VioletSlider CAPTCHA implementing CM method

Algorithm No. 4 (The CM Method)

1. Set *char_no* to 0 and state to *come*
2. Iterate each frame through *frames*
 - Iterate through *x* from 140 to 150
 - Set *cmp* to 0
 - Iterate through each *ix* from 0 to 2
 - Iterate through each *y* from 0 to 46
 - Set $offset = (frame_width * y) + (ix + x)$
 - If the frame pixel is not within *bg* pixel threshold at offset then
 - Increment *cmp*
 - If *cmp* is greater than 10
 - If state is *stay* and *x* is equal to 140 then
 - Set state to *leave*
 - Else if state is *leave* and *x* is greater than 140 then
 - Set state to *come*
 - If state is *come* then
 - Apply HS method just in *rect* from 140 to 180th column
 - Apply 'chunkFromWave' method to retrieve chunks



Fig. 5. The GBI method

Algorithm No. 5 (The GBI Method)

1. Create an empty image *BI* equal to width and height of the image.
2. Create an empty array.
3. Traverse the 15th row of *frames [1]* and add each pixel to the array.
4. Find the most used pixel from the array and assign it to variable *bg_color*
5. Create an empty array.
6. Traverse the 48th row of *frames [1]* and add each pixel to the array.
7. Find the most used pixel from the array and assign it to the variable *footer_color*
8. Traverse the 1st to 47th rows of *frames [1]* and set each pixel of *BI* to *bg_color* pixel.
9. Traverse the 48th to 60th rows and 75th columns of *frames [1]* and set each pixel of *BI* to *footer_color* pixel.
10. Return *BI*

RF (Remove Footer): Footers displaying the name of CAPTCHA can be found in all CAPTCHAs and RF method is applied to remove them. As the footer is located at fixed locations in all CAPTCHAs therefore easily removed. Experiments show it is located at the 48th row until the last row and these lines are changed into background colour. Algorithm

No. 6 outlines the steps for removing the footer from Hello CAPTCHAs. Fig. No. 6 depicts a clear background after the footer has been removed. *Bg_color* displays the actual background colour of the frame, and the *frames* argument contains the frames of the CAPTCHA.

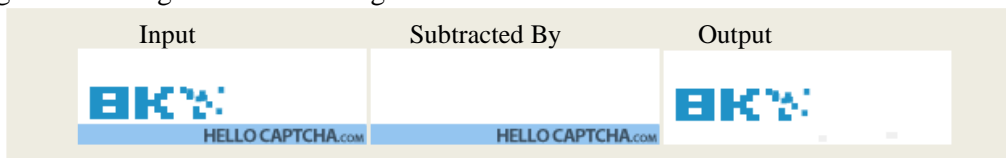


Fig. 6. An example of RF method

Algorithm No. 6 (The RF Method)

1. Set *firstframe* variable to the initial value in the *frames* array
2. Start a loop that iterates through each frame in the *frames* array
3. Set *j* to $(i+1)$ modulo *frames_length*
4. Set *frame* variable to the index *j* in the *frames* array
5. Iterates through each pixel in the frame starting from the 47th row
 - Set the frame pixel to $(bg_color + frame_pixel - firstframe_pixel) \% 256$
6. End the loop for each pixel after the 47th row
7. End the loop for each frame in the *frames* array

3.2 Removing Noise

GP (Grace Pixel): The process of going through all the pixels in an image is done using this method. Average of each pixel and its 8 adjacent pixels is calculated by dividing the sum by 9. This average pixel is then compared to the *grace* pixel by applying the *col_ratioidiff* method. Colours that are near each other are considered as zero, while those that are far from each other are assumed as close to 100. If the result of *col_ratioidiff* is less than the *grace_point*, then the *grace_to colour* is used to replace the colour

of the image at a certain location. Otherwise, the original colour of the image *img* is kept. This method is used to remove the noise that is surrounded by background pixels in Spread Fade/Braunie CAPTCHAs. Any remaining noise in the form of lines surrounded by background is also eliminated. An example can be seen in Fig. No. 7 and the steps are given in Algorithm No.7. *Img* argument is an input image which may have noise. *Grace* is output of this algorithm that is the image without noise.



Fig. 7. GP (Grace Pixels) method

Algorithm No. 7 (The GP Method)

1. Create an image *grace* of size *img width*, *img height*.
2. Iterate through each pixel from the image *img*:
 - a. Calculate the sum of the pixel and its surrounding 8 pixels, and divide it by 9
 - b. Compare the value calculated by *col_ratioidiff* function with *grace_point*
 - i. If the *col_ratioidiff* is less than *grace_point*, set *grace* pixel to *grace_to*
 - ii. Otherwise, set *grace* pixel to *img* pixel.
3. Return the image *grace*.

RC (Replace Colors): This method is used to substitute colours in an image. It examines all pixels in the image and compares them to each existing colour using threshold values. If the comparison is successful, it replaces the colour; otherwise, it is replaced with failure according to Algorithm No. 8.

This method is employed to solve Spread fade / JacobArmen types of CAPTCHAs, which have a near constant colour identified by CAPTCHA type. Moreover, this method also converts the image into binary format.

Algorithm No. 8 (The RC Method)

1. Iterate through each pixel in the *img*
 - a. check if the pixel is between the threshold of *cols*
 - i. If the condition is true, set the pixel to *success_col*
 - ii. If the condition is false, set the pixel to *failure_col*
2. End the loop

The *img* argument is a static image with the colours changed to either success or failure colours. The argument *cols* is used to choose colors from an array. The *threshold* argument is used to determine the amount of colour change that should take place. The *success_col* argument is used to replace the pixels between colour thresholds with a *success_color*. When the comparison does not succeed, the *failure_col* argument is used, and the pixels are replaced with this argument's colour.

BI (Binary Image): In this method, the RGB image is transformed directly into a binary image using the *mid_point* argument. The *mid_point* is

compared to a ratio which is calculated using either Equation No.1 (a) if the background is available, or Equation No.1 (b) if it is not.

$$ratio = (img_pixel - bg_pixl) / 255 \quad 1(a)$$

$$ratio = pixel / 255 \quad 1(b)$$

After *mid_point* is compared to the ratio, if the ratio is greater than *mid_point*, then the colour is changed to white, or RGB (255, 255, 255). If the ratio is less than *mid_point*, then the colour is changed to black, or RGB (0, 0, 0). This is demonstrated in Fig. No. 8 and outlined in Algorithm 9.



Fig. 8. An Example of Image Binarization

Algorithm No. 9 (The BI Method)

1. Iterate through each pixel of the image
 - a. Calculate the ratio of the pixel by dividing the pixel value by 255
 - b. If a background value is not null, find the absolute value of the gap between the pixel value and the background value, then divide by 255
 - c. If the ratio is greater than the *midpoint*; set the image pixel to RGB (255, 255, 255)
 - d. If the ratio is less than the *midpoint*, set the image pixel to RGB (0, 0, 0)

3.3 Segmentation

HS (Hierarchical Segmentation): The HS method is used to segment characters into small chunks, and the chunks are then stored in small images of 20*20 using the IFW (Image from Wave) method. The HS method involves two factors: *Wave* and *Hierarchy*. *Wave* is used to hit the target, creates several waves in the image using Algorithm No. 10 and the modified Flood Fill method. To ensure accuracy, the *wave* must meet certain specifications, as determined by the type of CAPTCHA, such as number of pixels, density, start position, width, and height. The waves will only segment the areas that have not already been segmented, and the process continues until the entire image is segmented.

The modified FloodFill technique is utilized in the HS method to act as a wave. In this algorithm, the data variable is in the form of a matrix, which allows a map view to store the wave index from the segmented areas of the current wave. Moreover, the other waves also store their wave index in the same variable, i.e. data. Once the entire image is segmented, the overall results of all segments are retrieved from the data variable and visualized as an image. In the modified FloodFill algorithm

(Algorithm No. 11), the wave's neighbouring waves are checked and saved in the 'wave' argument.

3.4 Predict Chunks

This method is used to convert static images into binary images, where the background is black, and the foreground is white. It scans for balls or strokes in the image, and if any are found, they are removed. Each pixel is stored using one byte of data, with 0 representing black (background) and 1 representing white (character). Therefore, a 20x20 pixel image requires 50 bytes of data, plus an extra byte for the label of the character, making a total of 51 bytes per pre-trained image. All characters from one CAPTCHA are saved in one pre-trained dataset, meaning that the dataset occupies 306 bytes (51*6). This method also converts background pixels to black and foreground pixels to white in order to create a binary image.

4. Breaking Hello CAPTCHAs

The Hello CAPTCHA is a free, user-friendly CAPTCHA system with an animated, text-based design that can be found online.

It is the most comprehensive set of animated CAPTCHAs available on the internet therefore we decided to verify its robustness.

Algorithm No. 10 (The HS Method)

1. Initialize stack *stk* and *waves* array
2. Iterate through *j* from 0 to *max_waves*
 - a. Check if stack is empty
 - If true then get index of data array where value is 0 and assign it to point variable
 - b. Push the point to the stack if point is greater than 0
 - c. From the stack, get the coordinates of the neighbours, pixels and *rect*
 - d. Perform FloodFill using the *img*, stack, data, *j*, and *wave*
 - e. Check if the constraint returns true
 - If constraint is true, set all entries in the data array with value equals to *wave_index* to -50
 - i. Pop the wave from *waves* array
 - ii. Decrement *j*
3. End the for loop
4. Create an empty array *hierarchy*
5. Call *WavesHierarchy* to form the hierarchy of waves
6. Return *data*, *waves*, and *hierarchy*

Algorithm No. 11(The MFF Algorithm)

1. While stack length is greater than 0
 - a. Pop stack and assign value to point
 - b. Set $data [point]$ equal to $wave_index$
 - c. Get pixel at point from img and assign it to p_col
2. Iterate through i from 0 to 9
 - a. Calculate $ix = (i\%3) - 1$ and $iy = \text{int}(i/3) - 1$
 - b. If $0 \leq point_x + ix \leq 179$ AND $0 \leq point_y + iy \leq 59$
 Check $data[(iy + Point_y) * frame_width + (ix + point_x)]$
 - c. If check is equal to 0 OR (check is less than 0 AND check is not equal to $-wave_index$) then
 - i. Calculate $new_point = (iy + point_y) * frame_width + (ix + point_x)$
 - ii. Assign the pixel from the img at new_point to col
 - d. If col is within threshold of p_col then
 - i. If yes, push new_point in stack
 - ii. Else, set $data [new_point] = wave_index$
 - e. If $check > 0$ AND check Not Equal To $wave_index$ then
 Set $wave_neighbours[check]=1$

The creators of the mentioned CAPTCHA systems assert that their designs are more functional, more secure, and visually appealing to draw in users. To make them more difficult to breach by automated means, various animation techniques are being considered. Even though the Hello CAPTCHA scheme was not designed to defend against segmentation attacks, it was created to be easily readable and understandable for humans. This study has chosen the Hello CAPTCHA scheme because it provides various animations that have been created by the designers. We believe that no other developer or website provides more animated text-based CAPTCHAs than HELLO CAPTCHAs. Each CAPTCHA challenge consists of six characters or numbers (image in GIF format) of 180*60 pixels. To break these animated CAPTCHAs, we have developed several algorithms, as mentioned earlier.

4.1 Flitter CAPTCHAs

In Flitter CAPTCHA, letters are formed from squares which are then scattered apart. Different assembly effects may be chosen, such as the erode effect that is the explosion with gravity. The designers of the Hello CAPTCHA used security strategies that involved distributing the information across multiple frames. There are seven varieties of Flitter CAPTCHAs:

- Flitter/ Bee CAPTCHAs
- Flitter/ Brick CAPTCHAs
- Flitter/ Default CAPTCHAs
- Flitter/ RustyFlam CAPTCHAs
- Flitter/ Vinayak CAPTCHAs

- Flitter/ Windowlicker CAPTCHAs
- Flitter/ colorful CAPTCHAs

Our proposed algorithms can successfully break the Flitter CAPTCHA and its seven subtypes with high precision. Initially, the background of the image is retrieved by employing GBI (Get Background Image), and the characters from frames are extracted using PCM (Pixel Changes Map). The proposed methods result high precision in breaking all six subtypes of the Flitter CAPTCHAs. Once the animated image has been loaded and its type has been identified, the GBI method can be implemented to obtain a clear background from the given image. The PCM method is used to extract the animated characters into a static image. Different types of noise need to be removed, and overlapped characters have to be segmented. The GP method is used to get rid of any noise in the image so that a clear picture can be produced. HS method is used to segment characters in different colours. Finally, The BI method is utilized to obtain characters in a binary form. Fig. 9 displays the results of all the varieties of the Flitter CAPTCHAs.

4.2 H-Mover CAPTCHA

H-Mover is an aesthetically pleasing CAPTCHA type, consisting of seven subtypes that display letters one by one from left to right in an animation where the letters come in from the right, stay still for a bit, and then depart to the right. The speed of the animation gradually increases and decreases, making it attractive. At no point in the animation are all of the letters visible simultaneously. There are seven varieties of H-Mover CAPTCHAs.

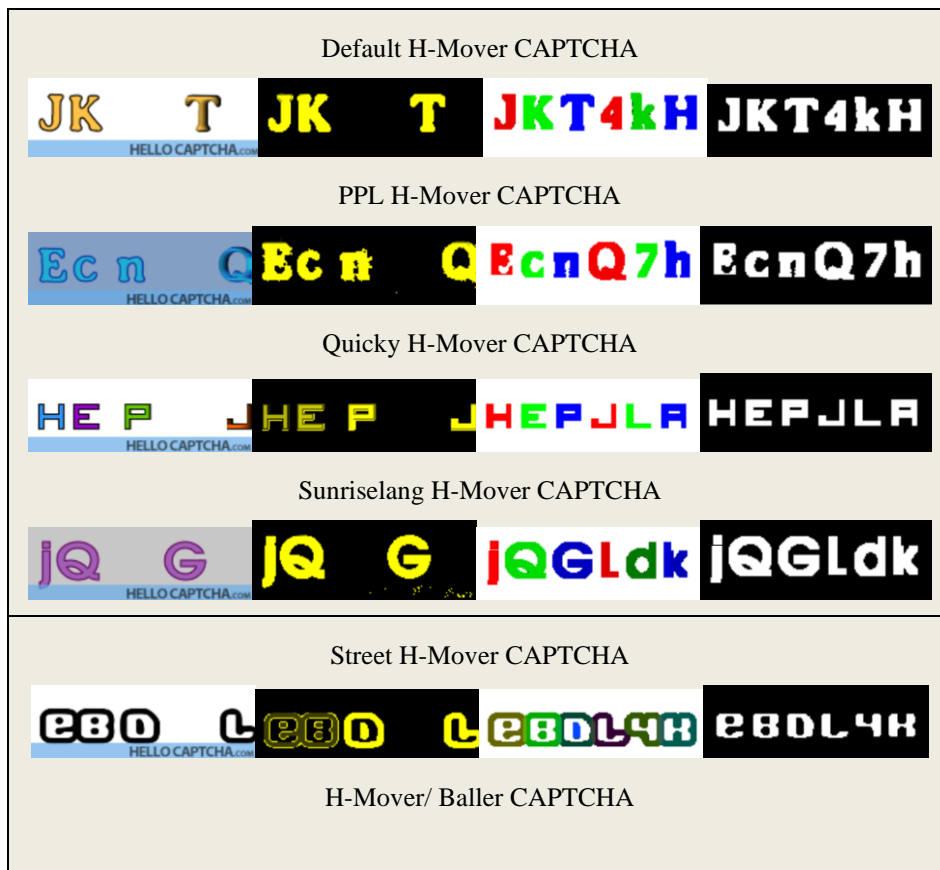
- H-Mover/ Default

- H-Mover/ PPL
- H-Mover/ Quicky
- H-Mover/ Street

- H-Mover/ Sunriselang
- H-Mover/ Violet slider
- H-Mover/ Baller



Fig. 9. Frames, Segmentation, Static and Binary image (From left to right)



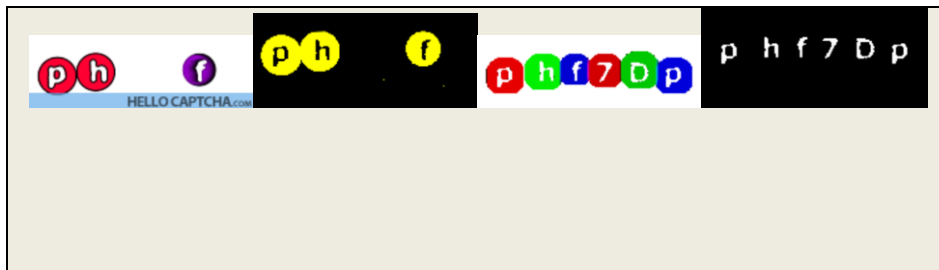


Fig.10. Extracting H-Mover CAPTCHAs

The proposed HS method in H-Mover/Sunriselang CAPTCHA is used to take animated characters from an image made up of various animated frames and turn it into a still image. As illustrated in Fig. 10, the segmented chunks are then retrieved.

4.3 Popup CAPTCHA

The letters appear in a random order, stay visible for a brief period of time, and then disappear. The colour of the letters, the speed of popup and pause time can all be altered. Popup CAPTCHAs have five different sub-types:

- Popup/ Day-rising
- Popup/ Default
- Popup/ Lazzari
- Popup/ Orange wave
- Popup/ Cirque

As shown in Fig. No. 11, the characters in a popup/cirque type of CAPTCHA have been condensed and extracted into a single image by scrubbing the image. The background and colours of

the image have been inverted to reveal the actual image.

4.4 Smarties CAPTCHAs

An attractive animation is used to display the characters from left to right in Smarties CAPTCHAs, with an extra offset parameter that allows them to start fading in earlier than waiting for the previous character to fade out. There are eight different sub-types of these CAPTCHAs.

- Smarties/Confronted CAPTCHA
- Smarties/ Cyan menace CAPTCHA
- Smarties/ Cyanmenace1 CAPTCHA
- Smarties/ GCG CAPTCHA
- Smarties/ Graphite CAPTCHA
- Smarties/ Palavollo CAPTCHA
- Smarties/ Smartie CAPTCHA
- Smarties/ Techware CAPTCHA

The results of this algorithm for Smarties CAPTCHAs and its sub-types are displayed in Fig. 12.

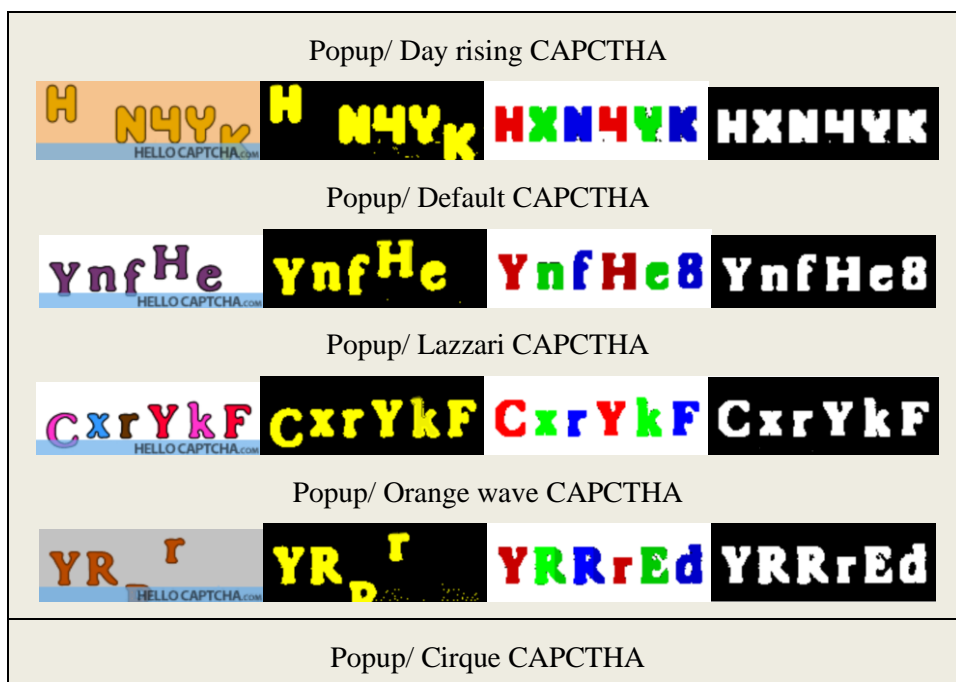




Fig. 11. Results of Popup CAPTCHAs

Smarties/ Confronted CAPTCHA



Smarties/ Cyan menace CAPTCHA



Smarties/ Cyanmenace1 CAPTCHA



Smarties/ GCG CAPTCHA



Smarties/ Graphite CAPTCHA



Smarties/ Palavollo CAPTCHA



Smarties/ Smartie CAPTCHA



Smarties/ Techware CAPTCHA



Fig. 12. Results of Smarties CAPTCHAs

4.5 SpreadFade CAPTCHA

The SpreadFade CAPTCHA has two layers: the text on the background layer and the squares on the foreground layer. Initially, the squares are visible; however, they slowly become transparent, making the letters visible. The edges of the fading can be altered by adding noise, making the edges wider and less precise. There are seven varieties of SpreadFade CAPTCHAs.

- SpreadFade/ Braunie

- SpreadFade/ Default
- SpreadFade/ Immature dream
- SpreadFade/ Jacob Armen
- SpreadFade/ Negative
- SpreadFade/ Ooo
- SpreadFade/ Taher

We can determine the width and height of each cell of an applied grid by saving the pixels in an array

after taking away the grid lines. From the pixels in the array, we create polygons and then draw all of them within a static image, as displayed in Fig. 13.

Although, one type of SpreadFade CAPTCHA, i.e. “Negative” falls in type 2, but it is not recognized by

our proposed algorithm, it is rather successfully recognized by our algorithm for SearchLight CAPTCHA.

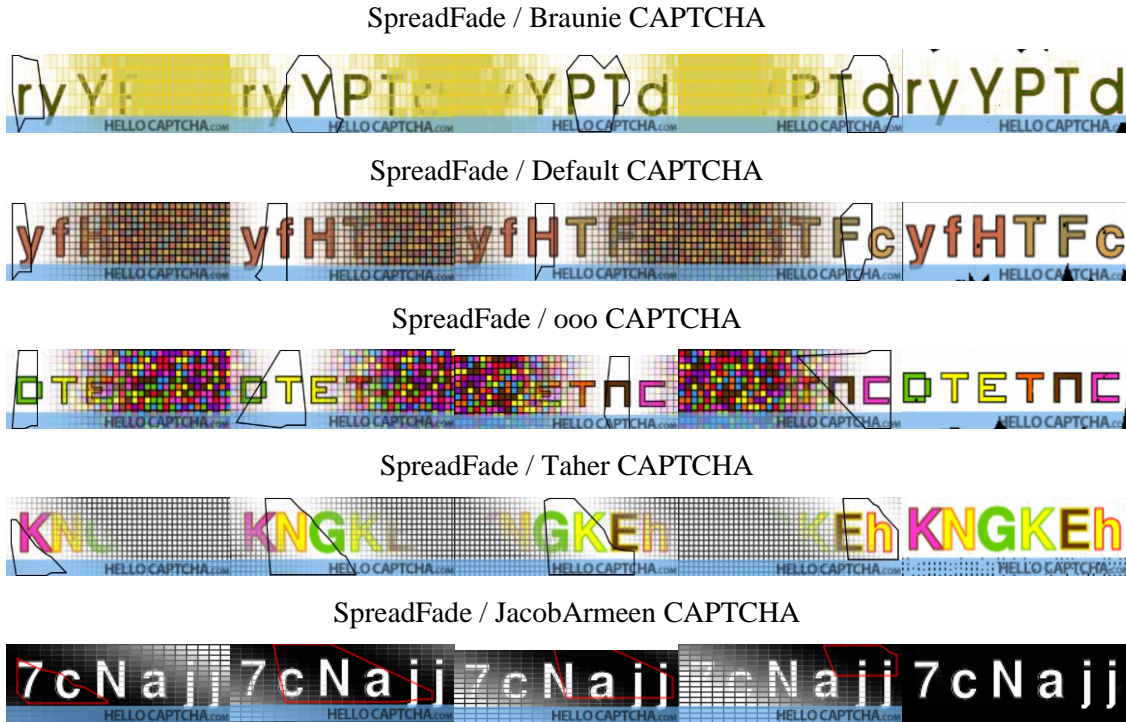


Fig. 13. The results of SpreadFade CAPTCHAs

4.6 SearchLight CAPTCHAs

These CAPTCHAs are characterized by a background colour and foreground text that appear so analogous that the text cannot be seen until a searchlight of a different hue is shined upon it. The text can only be seen when the searchlight precisely illuminates the foreground text as shown in Fig. 14. There are seven varieties of these CAPTCHAs:

- SearchLight/Aunt bloodCAPTCHAs
- SearchLight/ BuxstoreCAPTCHAs
- SearchLight/ CyberCAPTCHAs
- SearchLight/ GrünNinjaCAPTCHAs
- SearchLight/ MoanCAPTCHAs
- SearchLight/ RedlightCAPTCHAs
- SpreadFade / Negative CAPTCHA

The Swapper, Spring and Roller CAPTCHAs are the CAPTCHAs where individual characters are not fixed and they may have different locations, rotations and positions in every frame.

5. Results and Discussions

We have carried out various experiments to assess the accuracy of our proposed methods. We collected 4,100 samples of animated CAPTCHAs, with 100 samples of 41 different sub-types, to break Hello-CAPTCHAs. The table 1 illustrates the outcomes of our proposed method when used to break various schemes, including the Overall Precision, the method to break specific type of CAPTCHA and obtained chunks from animated images. We have achieved a high degree of accuracy in breaking animated CAPTCHA schemes. It takes an average of 3 seconds to break the six-character challenge of the scheme, which is the same amount of time it takes a normal human to solve it. The Overall Precision (OP) of the proposed algorithm is obtained by the Success Recognition Rate (SRR) of the classifier, the accuracy of segmentation, and the total number of characters in an image, which is calculated using Equation No. 2 [18]:

$$OP(\%) = SSR\% (SRR\%)^n \quad (2)$$

In a dataset of images, Segmentation Success Rate (SSR) is the number of characters correctly segmented. For example, if an algorithm is used to segment 500 characters into 100 images containing 6

characters each, the segmentation accuracy would be $500/600 = 0.833$ or 83.3%. The Success Recognition Rate (*SRR*) is determined by the accuracy of the

classifier, while n is the number of characters in an image.

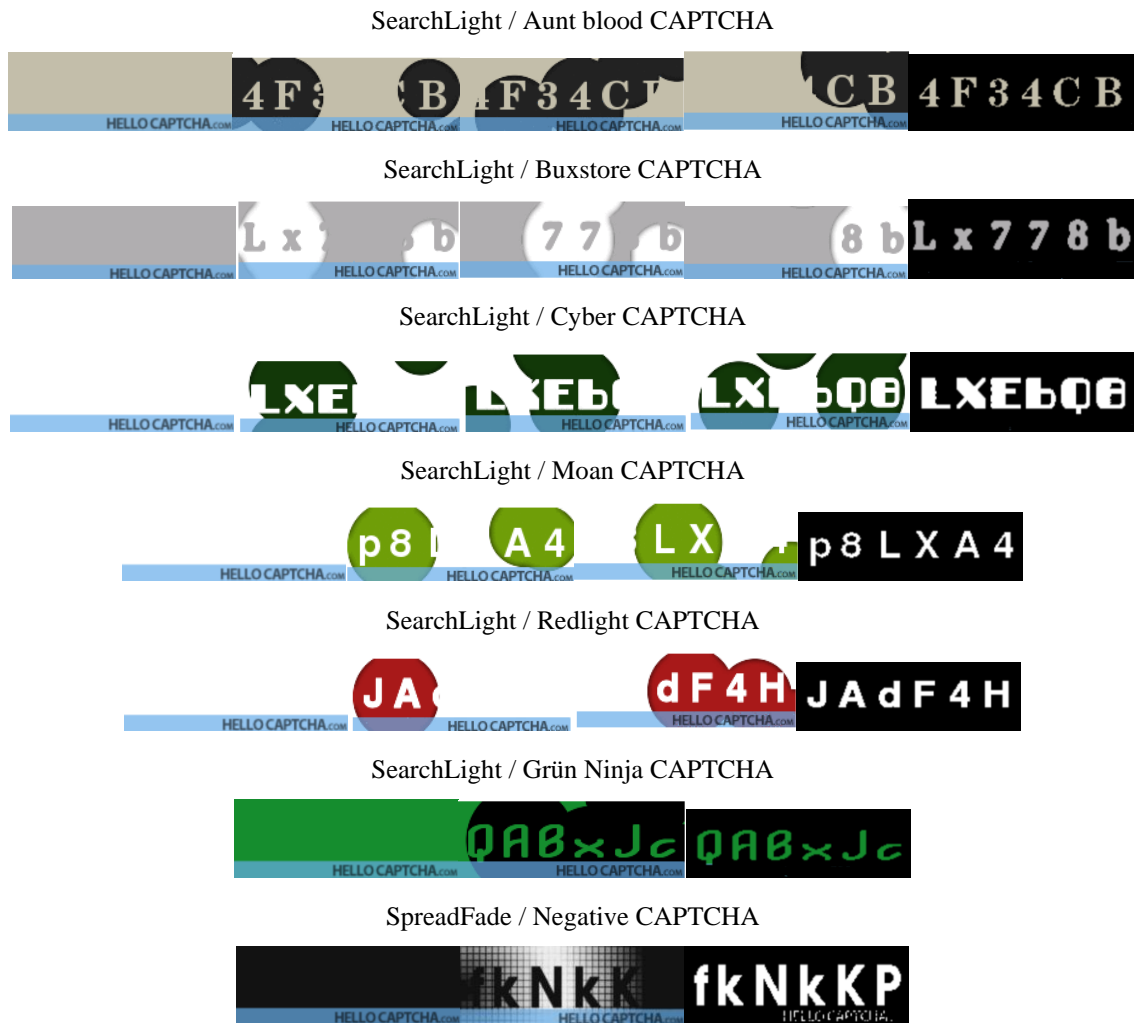


Fig. 14. The results of SearchLight CAPTCHA

Table 1

Results of all targeted HELLO-CAPTCHA schemes

S.No.	CAPTCHA Type	Method	Extracted Chunks of Characters	SSR%	SRR%	OP%
1	Flutter/Default	PCM	JXAN7B	100	98	99
2	Flutter/Bee	PCM	FCXLDP	100	98	99
3	Flutter/Brick	PCM	XBTXBR	98	98	98
4	Flutter/Rusty Flame	PCM	FKFAPQ	99	98	98
5	Flutter/Vinayak	PCM	R4KFBD	100	99	99.5
6	Flutter/Colorful	PCM	PRBECB	100	98	99

7	Flitter/ Windowlicker	PCM	PBTJFB	100	99	99.5
8	Search Light/ Aunt blood1	RCP	HKQPcx	100	98	99
9	Search Light/ Aunt blood	RCP	4F34CB	100	99	99.5
10	Search Light/ Cyber	RCP	Rea34Q	100	95	97.5
11	Search Light/ Grun Ninja	RCP	QABxJc	100	98	99
12	Search Light/ Redlight	RCP	JAdF4H	100	99	99.5
13	Search Light/ Moan	RCP	KxQpHC	100	98	99
14	SearchLight / Buxstore CAPTCHA	RCP	Lx778b	99	98	98.5
15	Spread Fade/ Taher	AF	FtDyx7	100	97	98
16	Spread Fade/ Immature Dream	AF	xJFQyP	100	98	99
17	SpreadFade/ JacobArmeen	AF	7cNa jj	99	99	98.5
18	Spread Fade/ooo	AF	DTKNLG	96	95	95.5
19	Spread Fade/ Negative	AF	pA3b7t	100	98	99
20	Spread Fade/ Default	AF	yfHTFc	99	98	98.5
21	SpreadFade/ Braunie	AF	ryYPTd	96	97	96.5
22	Smarties/Cyan Menace	PCM	TnHpnf	91	90	90.5
23	Smarties/ Confronted	PCM	D3hPGh	80	86	83
24	Smarties/Graphite	PCM	aRppYT	92	90	91
25	Smarties/ Techware	PCM	Fefh8c	80	86	83
26	Smarties/GCG	PCM	B3GQEN	90	92	91

27	Smarties/ Cyanmenace1	PCM	Pt 3xt 7	96	98	97
28	Smarties/Smarties	PCM	TCE4AN	90	92	91
29	Smarties/ Palavollo	PCM	F U 0 L b a	88	89	88.5
30	Popup/Default	PCM	YnfHe8	90	90	90
31	Popup/Cirque	PCM	C T B d E T	98	97	97.5
32	Popup/DayRising	PCM	HXN4YK	90	96	93
33	Popup/ lazzari	PCM	CxrYkF	90	92	91
34	Popup/Orange wave	PCM	YRRrEd	74	72	73
35	H-Mover/Defaul	CM	JKT4kH	88	89	88.5
36	H-Mover/ sunriselang	CM	JQGLdk	96	98	97
37	H-Mover/Baller	CM	p h f 7 D p	84	85	84.5
38	H-Mover/PPL	CM	EcnQ7h	100	82	81
39	H-Mover/Quicky	CM	HEPJLA	100	99	99.5
40	H-Mover/ VioletSlider	CM	FRLK4Y	98	98	98
41	H-Mover/Street	CM	EBDL4H	82	84	83

6. Conclusion and Future work

This paper introduces novel methods to decode the moving text in animated CAPTCHAs. In these CAPTCHAs, the information is distributed in several frames instead of a single static image. The idea is based on the assumption that by spreading it in various frames it is hard to be decoded by automated programs. However, in this work we have verified the robustness of most popular types of animated CAPTCHAs, i.e. Hello CAPTCHAs. We have attacked 41 types of Hello CAPTCHAs and successfully decoded them with high precision. We have proposed state of the art

methods to decode these CAPTCHAs. Novel image extraction, noise removal and image processing methods are applied to obtain the image containing required text. The obtained segments of characters may contain connected or non-connected characters which are separated using proposed segmentation methods. Finally, a pre-trained CNN is used to recognize the characters. The overall precision of up to 99.5 % was achieved in this work.

In future, we plan to verify the robustness of other types of animated CAPTCHAs as well as different design variants of them. We are also

looking forward to designing a new CAPTCHA in local language such as Urdu or Sindhi language which can be useful to protect the local websites where these languages are spoken and written. This idea can open a new research paradigm and challenge the research community to work on these languages which can ultimately improve the machine reading capabilities of said languages.

Declarations

Conflict of interest Authors declare that they have no conflicts of interest.

7. References

- [1] H. Gao, X. Wang, F. Cao, Z. Zhang, L. Lei, J. Qi and X. Liu, "Robustness of text-based completely automated public Turing test to tell computers and humans apart", *IET Inf. Secur.* 10(1), 45–52, 2016. <https://doi.org/10.1049/iet-ifs.2014.0381>
- [2] E. Bursztein, M. Martin and J. Mitchell, "Text-based CAPTCHA strengths and weaknesses", *Proceedings of the 18th ACM conference on Computer and communications security.* 125–138, 2011. <https://doi.org/10.1145/2046707.2046724>
- [3] H. Gao, W. Wang, Y. Fan, J. Qi and X. Liu: The Robustness of Connecting Characters Together CAPTCHAs. *J Inf Sci Eng* 30(2), 347–369, 2014.
- [4] Creo.Group.: HelloCAPTCHA vs Spam bots, 2012. Available at: <http://www.hellocaptcha.com> (Accessed: 20 January 2023)
- [5] Y. W. Chow and W. Susilo, "AniCAP: An animated 3D CAPTCHA scheme based on motion parallax", *International Conference on Cryptology and Network Security*, Springer. 255–271, 2011.
- [6] V. D. Nguyen, Y. W Chow and W. Susilo, "Attacking animated CAPTCHAs via character extraction. *International Conference on Cryptology and Network Security*", Springer. 98–113, 2012. https://doi.org/10.1007/978-3-642-35404-5_9
- [7] R. Hussain., H. Gao. and R. A. Shaikh, "Segmentation of connected characters in text-based CAPTCHAs for intelligent character recognition", *Multimedia Tools and Applications.* 76(24). 25547–25561, 2017. <https://doi.org/10.1007/s11042-016-4151-2>
- [8] Y. W. Chow, W. Susilo and P. Thorncharoensri, "CAPTCHA design and security issues. *Advances in cyber security: principles, techniques, and applications*", Springer. 69–92, 2019. https://doi.org/10.1007/978-981-13-1483-4_4
- [9] E. Athanasopoulos and S. Antonatos, "Enhanced captchas: Using animation to tell humans and computers apart", *IFIP International Conference on Communications and Multimedia Security*, Springer. 97–108, 2006. https://doi.org/10.1007/11909033_9
- [10] J.S. Cui et al. "A CAPTCHA implementation based on moving objects recognition problem", in 2010 *International Conference on E-Business and E-Government IEEE.* 1277–1280, 2010. <https://doi.org/10.1109/ICEE.2010.326>
- [11] K. Kunitani and R. Uda, "Proposal of CAPTCHA using three dimensional objects", *Proceedings of the 7th International Conference on Ubiquitous Information Management and Communication.* 1–6, 2013. <https://doi.org/10.1145/2448556.2448613>
- [12] Y.W. Chow and W. Susilo, "AniCAP: An animated 3D CAPTCHA scheme based on motion parallax", *International Conference on Cryptology and Network Security*, Springer. 255–271, 2011.
- [13] Inc., L. M. T., "NuCAPTCHA", Available at: <http://www.nucaptcha.com>, 2013.
- [14] I. Kund, "Non-standard captchas for the web: a motion based character recognition hip", *Dissertation University of Monchester.* Jeff Yan, Ahmad Salah EI Ahmed Usability of CAPTCHAs or Usability issues in CAPTCHA design school of computing Science, Newcastle University, UK, 2011.
- [15] X. Xu. et al., "Characteristic analysis of Otsu threshold and its applications", *Pattern recognition letters*, Elsevier. 32(7), 956–961, 2011. <https://doi.org/10.1016/j.patrec.2011.01.021>
- [16] V. D. Nguyen, Y.W. Chow and W. Susilo, "Breaking a 3D-based CAPTCHA scheme. *International Conference on*

- Information Security and Cryptology”, Springer, 391–405, 2011. https://doi.org/10.1007/978-3-642-31912-9_26
- [17] J. Nian., P. Wang, H. Gao, and X. Guo, “A deep learning-based attack on text CAPTCHAs by using object detection techniques”, IET Information Security. 16(2), 97-110, 2022. <https://doi.org/10.1049/ise2.12047>
- [18] O. Starostenko, C. Cruz-Perez, F. Uceda-Ponga, and V. Alarcon-Aquino, “Breaking text-based CAPTCHAs with variable word and character orientation”, Pattern Recognition 48(4), 1101-1112, 2015. <https://doi.org/10.1016/j.patcog.2014.09.006>
- [19] M. Kumar, M. K. Jindal, and M. Kumar, “A systematic survey on CAPTCHA recognition: types, creation and breaking techniques”, Archives of Computational Methods in Engineering, 29(2), 1107-1136, 2022. <https://doi.org/10.1007/s11831-021-09608-4>
- [20] N.T Dinh and V.T. Hoang, “Recent advances of Captcha security analysis: a short literature review”, Procedia Computer Science. 218, 2550-2562, 2023. <https://doi.org/10.1016/j.procs.2023.01.229>
- [21] M. Guerar, L. Verderame, M. Migliardi, F. Palmieri and A. Merlo: GottaCAPTCHA’Em all, “A survey of 20 Years of the human-or-computer Dilemma. ACM Computing Surveys”, (CSUR). 54(9), 1-3, 2021. <https://doi.org/10.1145/3477142>